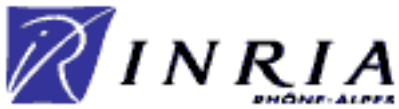
[Tardif's home page]



INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

THESE

pour obtenir le grade de

DOCTEUR DE L'INPG

***Spécialité :* "*Informatique système et communication*"**

préparée au laboratoire INRIA Rhône-alpes

dans le cadre de **l'ECOLE DOCTORALE** " ***Mathématiques, sciences et technologie de l'information***"

présentée et soutenue publiquement

par

Laurent Tardif

le 18/12/2000

Titre :

Kaomi : réalisation d'une boîte à outils pour la construction d'environnements d'édition de documents multimédias

Directeur de Thèse :

Mme Cécile Roisin

JURY

Mme Christine Collet	, Président
Mr Marc Nanard	, Rapporteur
Mr François Pachet	, Rapporteur
Mme Cécile Roisin	, Directeur de Thèse
Mme Muriel Jourdan	,Co-encadrant
Mr Jean-Pierre Peyrin	, Examinateur

Remerciements

Je tiens à remercier Cécile Roisin et Vincent Quint pour m'avoir ouvert les portes du projet Opéra voici quelques années.

Je remercie Cécile Roisin et Muriel Jourdan pour m'avoir guidé et supporté durant cette thèse.

Je tiens à remercier les membres du jury, Mmes Christine Collet, Cécile Roisin, et Muriel Jourdan, ainsi que Mrs Marc Narnard, François Pachet et Jean-Pierre Peyrin.

Je remercie Cécile et Muriel qui ont eut la tâche ingrate de relire cette thèse.

Je remercie aussi tout les stagiaires avec qui j'ai eu l'occasion de partager et d'échanger de nombreuses idées, et j'encourage tous les doctorants ainsi que tous les chercheurs à encadrer des stagiaires pendant leur travail.

Je tiens à remercier tous ceux ,qui, un jour ou l'autre, ont développé au sein de Kaomi et des différents environnements auteur.

Je voudrais aussi remercier tous les collègues passés et présents du projet qui par leurs conseils, leurs encouragements et leur aide ont contribué à l'aboutissement de cette thèse. Je remercie les belges (Christophe et Miguel), les Ziut (Thomas, Régis), les maîtres (Victor, Michael, Stéphane), les DEA (Marion), les (ex) cnam (Laurent, Maximilien et Patrice), les (ex) thésards (Loay, Nabil, Fred et Fred, Lionel, Tien, Stephane) et enfin tous les pongistes de l'INRIA.

Je tiens aussi à remercier Alain G. et Eric R. qui ont su me faire découvrir d'autres aspects de la thèse.

Je remercie tous mes amis (Lionel, Fred^2, Yann & Sylvie, Sylvain, Pierrot, Teorem) qui ont toujours été de bons conseils, et enfin je remercie l'ange de ma vie pour m'avoir supporté et encouragé pendant ces années.

Table des matières

Chapitre I : Introduction

Depuis une décennie, les progrès en informatique sont tels que le monde du multimédia a pris un essor considérable. Avec l'avènement du Web, l'évolution du matériel, les besoins en traitement et en diffusion de l'information sont de plus en plus importants. De nombreux travaux portent aujourd'hui autour de ces thèmes. Ils concernent la création de périphériques adaptés à la présentation de ces informations, la définition de réseaux haut débit pour transporter l'information ou la spécification de standards pour encoder l'information et favoriser ainsi les échanges entre les personnes.

Mais, paradoxalement, peu de travaux portent sur *l'édition* de ces informations. L'édition consiste à agencer et intégrer un ensemble de données numériques, appelées *médias*, de nature diverse (son, vidéo, image,). Un tel processus est nécessaire lorsqu'une personne souhaite créer ou télécharger des médias puis les assembler pour écrire, par exemple, une page Web. De la même manière, un institut ou une entreprise qui veut utiliser le Web pour diffuser de l'information sera confronté à une phase d'édition. On comprend alors que dans de nombreuses situations, l'information doit être structurée et mise à jour régulièrement.

Avec l'apparition d'informations de plus en plus riches, les documents qui les contiennent deviennent de plus en plus complexes et difficiles à spécifier. De documents purement textuels il y a quelques années, on est arrivé aujourd'hui à des documents interactifs et temporisés. Par la suite, nous appellerons *document multimédia* tout document composé de médias de différents types, et dont la présentation comporte une composante spatiale, hypertexte mais aussi temporelle. Il devient donc indispensable que l'auteur soit aidé dans cette tâche d'édition par un logiciel. Ces logiciels seront appelés par la suite *environnements auteur* ou *environnements d'édition*.

Afin de rendre la création de documents multimédias accessible à tous, il est nécessaire de proposer des *environnements auteur* permettant à tout un chacun (sans supposer de compétences informatiques particulières) d'éditer des documents multimédias.

2. Conception et édition de documents multimédias

Le processus d'élaboration d'un document multimédia que ce soit dans un contexte professionnel ou familial suit les mêmes étapes [Bollard99], même si certaines de ces étapes sont plus ou moins complexes et développées en fonction du contexte. Par exemple, si nous faisons le parallèle avec l'écriture de documents HTML, dans un contexte familial l'auteur crée empiriquement la structure de son document alors que dans le cas d'un site professionnel cette structure est formalisée préalablement.

De façon générale, le processus de conception d'un document multimédia nécessite la participation de plusieurs personnes : les personnes du marketing qui décident du contenu à mettre en avant, les graphistes qui réalisent les illustrations, et enfin les techniciens qui mettent en oeuvre le document.

Plus précisément, ce processus de conception peut être décomposé en sept étapes :

- *Apparition d'une idée* de document à produire ou commande d'un document.
- *Clarification de cette idée :* à partir de l'idée initiale il faut dégager des informations comme les objectifs de la présentation, le type de public concerné, les fonctionnalités attendues, le choix de support de communication utilisé. Un *synopsis* permet de mettre en forme l'ensemble des réponses aux questions précédentes.
- *Ecriture du scénario* : une fois que l'on a décrit l'objectif du document, il faut préciser les moyens et les méthodes pour y arriver : quelle stratégie utiliser pour l'exploration des informations ou la navigation dans le document.
- *Ecriture du scénarimage* (story-board) : il décrit de manière visuelle le résultat graphique attendu. Il permet d'identifier les médias utilisés ainsi que l'interactivité à attacher à ces médias.
- *Réalisation de la maquette* : à la fin de l'écriture du scénarimage, on réalise une maquette qui permet de tester un scénario et d'avoir une première idée du document dans sa forme finale. À partir de cette maquette, on peut choisir de remettre en cause des choix faits précédemment. Actuellement dans un contexte commercial la réalisation de cette maquette est laissée aux soins des informaticiens et nécessite une description très précise du scénarimage et du scénario.
- *Production du document*: une fois les choix faits, il ne reste plus qu'à produire le document. Cette phase se décompose en deux sous-tâches :
  - Création ou collecte des médias qui composeront le document, cette phase est généralement réalisée par des graphistes.
  - Assemblage des différents médias pour constituer le document.
- *Diffusion*: une fois le document produit, il est diffusé. Cette diffusion peut être réalisée par l'intermédiaire de CDROM ou via le Web.

Au cours de cette thèse nous nous intéresserons essentiellement aux trois dernières étapes de ce processus, c'est-à-dire à la production du document proprement dite. Nous n'oublierons cependant pas, qu'au cours de cette production, des remises en cause du scénario ou du scénarimage peuvent arriver à tout moment. Un des objectifs de la thèse sera de rendre accessible la réalisation de la maquette à des non-informaticiens et plus particulièrement aux personnes qui écrivent le scénarimage.

3.Représentation d'un document multimédia dans un contexte d'édition

Avant d'aller plus loin dans cette thèse nous allons décrire les différentes représentations que peut prendre un document au cours du processus d'édition :

- *La structure interne du document*: elle représente le document au sein de l'environnement auteur. Le système d'édition permettra à l'auteur de manipuler cette structure de données au travers de fonctions d'édition. Nous appellerons par la suite *formalisme d'édition* l'ensemble des informations que l'auteur perçoit de la structure interne du document.
- *Format de sauvegarde*: il permet de stocker le document dans un fichier. Ce format peut être un format propriétaire du système auteur (Madeus [Layaïda97], Director [Macromédia-Director00]) ou il peut être un des standards de sauvegarde (SMIL [SMIL98]). Ce fichier contient les informations liées à l'édition du document et à l'environnement auteur, mais il peut aussi contenir les informations de présentation du document.
- *Format de restitution*: représente le document dans une (ou plusieurs) forme(s) distribuée(s)

aux lecteurs. Dans le cas d'une diffusion pour plusieurs périphériques (téléphones mobiles, ordinateurs, assistants personnels) deux types de distribution sont possibles :

- la première solution consiste à distribuer un fichier par périphérique, c'est la solution communément utilisée aujourd'hui ;
- la deuxième solution consiste à fournir un fichier qui contient les informations pour plusieurs périphériques, on dit dans ce cas, que le document est *adaptable*.

Le fichier de restitution peut être :

- Un programme directement exécutable par le système d'exploitation de la machine ou interprétable par une machine virtuelle (Java).
- Un fichier interprétable par un logiciel spécifique :
  - Un fichier textuel, écrit dans une forme déclarative (HTML, SMIL, Madeus) ou non (Postscript). L'émergence de standards comme XML permet d'homogénéiser la manière de décrire des structures de données et en particulier des documents. Ceci afin de faciliter le partage des données.
  - Un fichier binaire (AVI, MPEG).

Ce fichier de présentation doit contenir les informations de présentation, mais il peut aussi contenir les informations d'édition qui ont mené à cette présentation de manière à permettre la réédition.

Equivalence de représentation

Par la suite nous dirons que deux représentations du document sont *équivalentes* pour l'édition, si l'auteur a le même pouvoir d'édition avec les deux représentations. C'est-à-dire qu'il peut réaliser la même opération d'édition élémentaire (ajout d'objet, changement d'attribut) sur les deux représentations, tout en obtenant la même présentation finale. Dans le cadre des logiciels propriétaires les trois représentations sont équivalentes et/ou confondues, c'est le cas du format interne de Word et du format de fichier "*.doc*". Par contre, un utilisateur de Word qui a exporté son document en HTML n'aura pas le même pouvoir d'édition suivant qu'il édite le document au format HTML ou le document au format Word. En effet, s'il insère une figure au début de son document Word, tous ses numéros et ses références de figures seront mis à jour, ce qui n'est pas le cas s'il édite son document HTML. En effet, lors de l'export du document Word en HTML, il a perdu des informations propres à l'édition. Dans ce cas on dit qu'il n'y a pas d'équivalence entre le format de sauvegarde (*".doc"*) et le format de restitution (HTML).

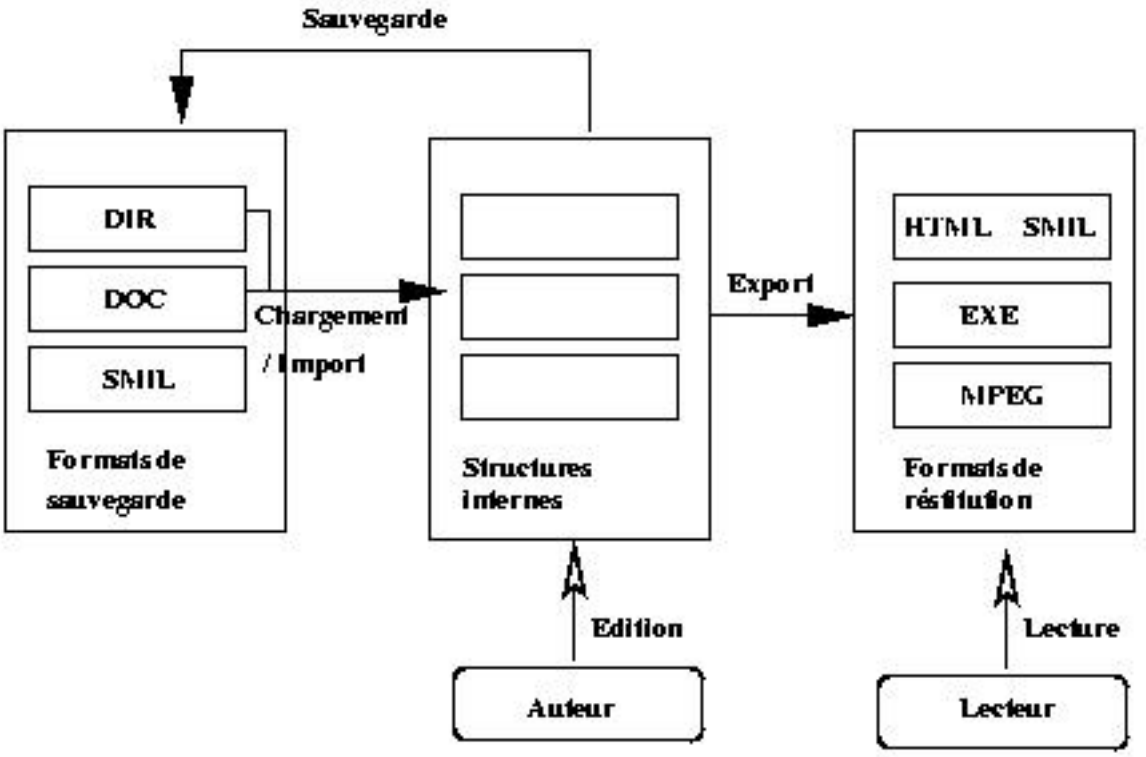


Figure I-1 : les différentes représentations physiques d'un document

Dans la Figure I-1, on peut voir l'enchaînement de ces différentes représentations au cours de l'édition. Dans la majorité des environnements auteur actuels (Director [Macromedia00c], ICON-Author [AimTech96]), il y a une équivalence entre les deux premières représentations mais pas avec la troisième. L'auteur, au moment de distribuer son document, *l'exporte* vers un format de restitution, perdant ainsi certaines informations liées au processus d'édition et donc, perdant aussi l'équivalence avec les structures précédentes. Les deux inconvénients majeurs de cette approche sont :

- la limitation de l'édition à un seul outil, du fait que les formats de sauvegarde manipulés sont souvent propriétaires, ce qui rend l'échange de données (avec conservation du potentiel d'édition) limité. De plus, la pérennité du document est très liée à celle de l'environnement auteur.
- La version distribuée aux lecteurs ne contient plus les informations permettant une réédition. De ce fait, l'auteur doit maintenir deux versions de son document, la version qu'il peut éditer, et la version qu'il distribue.

Ces limitations étant très contraignantes, certaines approches (CMIFed [Hardman93], Madeus-97 [Layaïda-97]) ont essayé de maintenir une équivalence entre les trois entités de représentation du document multimédia. L'intérêt majeur est de permettre une distribution du document tout en gardant le potentiel d'édition. L'émergence de nouveaux standards de présentation (par exemple SMIL) fait penser qu'il sera possible de faire cohabiter l'exigence d'avoir une pérennité dans l'édition et celle d'avoir un format de restitution normalisé. Cela permettra par exemple, une réédition du document avec un autre environnement auteur que celui qui a permis l'édition initiale du document.

Le rôle d'un environnement d'édition est donc d'offrir des représentations les plus proches possibles les unes des autres pour permettre une édition simple tout en favorisant la pérennité du document. Au cours du processus d'édition, les auteurs pourront manipuler et éditer chacune de ces représentations. Dans chacune de ces représentations on peut distinguer deux entités : une entité donnée (ou contenu) et une entité présentation (ou composition).

- *Donnée* : elle comprend différents médias qui peuvent être statiques ou dynamiques, structurés (SMIL, HTML) ou non. La collecte ou la création de ces médias se fait lors de la phase de production du document.
- *Présentation* : cette entité définit les interactions des différents médias les uns avec les autres ainsi qu'avec l'utilisateur. Cette entité comprend la synchronisation spatiale, temporelle et hypermédia mais aussi les interactions du lecteur avec le document. Lors du processus d'édition ces informations sont initialement décrites dans le scénario et le scénarimage avant d'être traduites dans le langage utilisé lors de la phase de production.

Dans le cadre de cette thèse nous nous placerons essentiellement dans le cadre de l'édition des informations de présentation du document.

4. Contexte de la thèse

Cette thèse s'est déroulée au sein du projet OPERA de l'INRIA Rhône-Alpes. Cette équipe travaille depuis de nombreuses années dans le cadre de l'édition de documents. Des travaux ont été menés selon plusieurs axes de recherche :

- *L'édition structurée* ([Quint87], [Bonhomme99]), dans laquelle il y a une distinction très forte entre le contenu et la présentation du document. Cela permet d'avoir plusieurs présentations pour un même document. L'éditeur Thot a permis de valider différents paradigmes d'édition. Pour cela, il fournit une édition structurée, cela signifie, par exemple, que chaque élément d'un texte est typé (liste, paragraphe, titre, sous-titre) et qu'il existe des règles de composition entre ces différents types comme le fait qu'il n'y ait pas de titre dans une liste. L'éditeur utilise ce typage pour présenter le document mais aussi pour faire un ensemble de vérifications automatiques (les listes ont au moins deux éléments, les sous-titres sont précédés de titres,). De plus, l'éditeur Thot [Quint99] permet à l'auteur de définir un modèle de documents indépendant de la présentation, ainsi que différents modèles de présentation pour ce modèle. Cela permet par exemple de définir un modèle de "référence bibliographique", et de définir plusieurs mises en page pour celui-ci.
- *L'édition coopérative*, dans laquelle l'intérêt est de permettre à plusieurs auteurs de collaborer pour écrire des documents. Les éditeurs Alliance [Decouchant96] et Byzance [Byzance00] permettent par exemple une telle coopération dans l'édition de documents. Le premier a été réalisé dans le cadre de l'édition de documents structurés, le deuxième dans le cadre de l'édition hypertexte.
- *L'édition multimédia*, dans laquelle l'objectif est d'intégrer une dimension temporelle aux documents. Les thèses de Nabil Layaïda [Layaïda97] et Loay Sabry-Ismail [Sabry99] ont permis dans un premier temps de définir un modèle d'expression de documents multimédias riche et souple, et, dans un deuxième temps, de réaliser un environnement auteur appelé Madeus-97 validant ce modèle et intégrant des fonctions de présentation. Notons que l'effort dans ces deux thèses a surtout porté sur la définition du modèle de document ainsi que sur les services de présentation et peu sur les services d'édition.

Ceci explique pourquoi à partir de ces travaux nous avons voulu mener une réflexion générale sur les problèmes liés à l'édition de documents multimédias.

5. Objectifs de la thèse

Le premier objectif de la thèse sera donc de dégager les besoins de l'auteur au cours du processus d'édition et d'apporter une réponse à ces besoins sous la forme d'une spécification détaillée des fonctions nécessaires dans un environnement auteur de documents multimédia s.

La volonté de valider ces idées nous a amené à avoir une réflexion sur l'architecture d'un environnement d'édition et à proposer une boîte à outils pour faciliter la création d'environnements auteur.

Le deuxième objectif de la thèse sera de concevoir et réaliser cette boîte à outils puis de la valider au travers de la réalisation de différents environnements auteur.

6. Plan de la thèse

Cette thèse est organisée de la manière suivante :

Dans le chapitre II, je présenterai dans un premier temps une classification des différents besoins d'un auteur de documents multimédias. Dans un deuxième temps, je présenterai le cycle d'édition d'un document multimédia. Je me baserai sur cette classification et sur ce cycle pour présenter et analyser les différents formats permettant de spécifier de tels documents. Dans une troisième étape, je présenterai les différents outils d'édition existants aujourd'hui.

Dans le chapitre III, je ferai une proposition d'environnement d'édition idéal permettant à l'auteur de suivre le cycle d'édition proposé dans le chapitre II. La description de cet environnement auteur se fera en plusieurs étapes. Pour chacun des besoins identifiés dans le chapitre précédent j'essayerai d'apporter une réponse pertinente en m'inspirant de ce qui existe aujourd'hui dans différents domaines liés ou non à celui du multimédia.

Dans le chapitre IV, je présenterai Kaomi. Kaomi est une boîte à outils qui permet de construire des environnements auteur de documents multimédias conformes aux idées décrites dans le chapitre précédent. Je présenterai l'architecture de cette boîte à outils ainsi que les différents services qu'elle offre.

Dans le chapitre V, je m'intéresserai plus particulièrement aux mécanismes sous-jacents qui ont permis de réaliser cette boîte à outils. Les techniques utilisées sont essentiellement des techniques à base de contraintes. L'objectif de ce chapitre sera de trouver un mécanisme de résolution de contraintes adapté à nos besoins. Pour cela, je présenterai différentes techniques permettant de résoudre des spécifications à base de contraintes. Je ferai cette analyse en fonction du contexte dans lequel j'utiliserai ces techniques. Je présenterai ensuite l'intégration de ces techniques dans Kaomi et l'évaluation des différents résolveurs de contraintes dans le cadre de l'édition de documents multimédias. Je profiterai de cette analyse pour faire part de certaines réflexions prospectives sur l'utilisation possible de ces techniques et de leur apport dans un contexte comme le multimédia qui reste en constante évolution.

Dans le chapitre VI, je présenterai les différents environnements auteur construits à partir de Kaomi. Je conclurai ce chapitre en faisant un bilan des différentes expériences d'utilisation de Kaomi.

Enfin, en conclusion, je tirerai un bilan de ces trois années de thèse et je donnerai quelques pistes de recherche.

Afin de faciliter la lecture, je précise que l'état de l'art se retrouve à trois niveaux :

- dans le chapitre II pour l'analyse des besoins de l'auteur ;
- dans le chapitre III pour l'étude des besoins en visualisation ;
- dans le chapitre V pour les mécanismes à base de contraintes.

et mes contributions se trouvent à quatre niveaux :

- la spécification d'un environnement auteur idéal, dans le chapitre III ;
- la réalisation de la boîte à outils Kaomi présentée dans le chapitre IV ;
- l'évaluation et l'implantation de résolveurs de contraintes adaptés aux besoins des auteurs de documents multimédias présentée dans le chapitre V ;
- la participation à la réalisation de différents environnements auteur présentés dans le chapitre VI.

Chapitre II : L'édition de documents multimédias

1.Définition du processus de création d'un document multimédia

Le processus de création de documents multimédias est un processus complexe qui ne fait pas forcément intervenir un environnement auteur. On peut distinguer trois principales manières de créer un document multimédia (illustrées par la Figure II-1) :

1. Edition via un environnement auteur : l'outil offre un support à l'auteur via une interface graphique et un ensemble de services, par exemple :

- Visualisation du document ou de certains aspects du document en cours de construction, en offrant différentes vues : vue de la structure, vue de l'organisation temporelle.
- Indépendance de l'auteur vis-à-vis de la syntaxe du langage dans lequel sera sauvegardé le document. C'est le cas par exemple des éditeurs de documents au format HTML (Netscape, Amaya) qui permettent à l'auteur d'écrire des documents sans se soucier de la syntaxe de ce langage.
- Automatisation de certaines tâches : maintenance des liens, vérification de la cohérence.

2. Edition via un éditeur de texte: ce type d'édition est encore souvent utilisé car les auteurs ne trouvent pas d'environnement d'édition qui satisfassent pleinement leurs besoins.
3. Génération automatique de documents: dans ce type de processus, l'intervention de l'auteur est minime voire inexistante. La production du document se fait par génération à partir d'un programme. Ce sont par exemple :

- Les documents construits pour présenter au lecteur le résultat d'une requête à une base de données. Le système doit alors ordonner les résultats et les présenter au lecteur.
- Les documents qui sont produits par traduction d'autres documents.

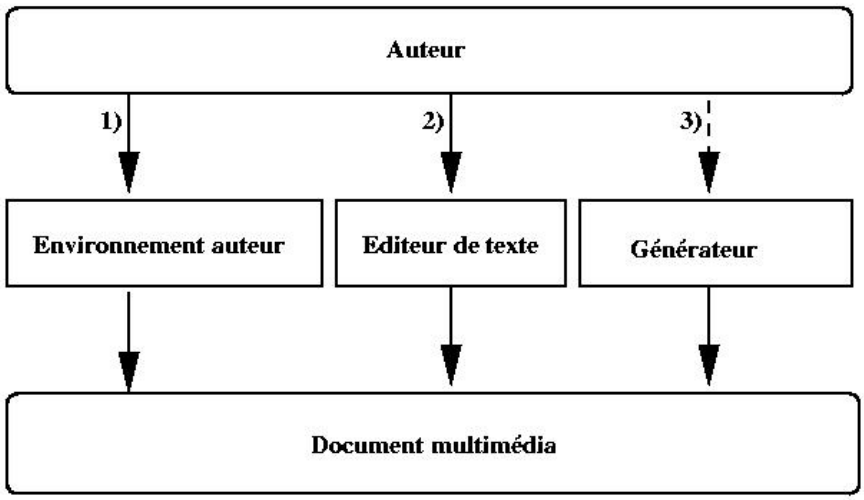


Figure II-1 : Les trois principaux modes de création de documents multimédias

Les générateurs de documents utilisent aujourd'hui la production automatique de documents. A partir d'un ensemble de données le générateur produit des informations de présentation pour permettre aux lecteurs de

les visualiser. Ce mode de création n'est pas adapté à une édition interactive.

L'objectif de cette thèse est d'apporter une réponse pertinente au problème de l'édition interactive de documents multimédias en se plaçant dans le contexte d'un environnement auteur. Nous espérons ainsi éviter le plus possible l'écriture de documents multimédias par l'intermédiaire d'éditeurs textuels. En particulier, il convient de porter une attention particulière au fait qu'avec l'émergence d'XML, de nombreux langages apparaissent. Il est donc nécessaire d'imaginer des environnements auteur qui soient capables de gérer cette multiplicité de formats.

L'objectif de ce chapitre est tout d'abord d'identifier les besoins des auteurs de documents multimédias et de confronter ces besoins aux langages et outils existants. La difficulté de ce travail vient de l'interdépendance entre langages et outils comme nous le verrons dans l'analyse de ces différents éléments.

Je commencerai donc dans la section 2 de ce chapitre par identifier les différents besoins issus du processus de création d'un document multimédia. Le but est de les confronter dans les deux sections suivantes, d'une part aux différents types de langage qui existent (section 3) et d'autre part aux outils auteur les plus représentatifs du domaine (section 4). Enfin, en conclusion de ce chapitre, je ferai un bilan des différentes approches de spécification et d'édition de documents multimédias.

2. Analyse des besoins pour la création multimédia

Dans le cadre des environnements auteur, l'auteur manipule indirectement le langage de sauvegarde. La structure de donnée interne de l'application est une représentation de ce fichier, représentation dont le rôle est de simplifier la tâche d'édition de l'auteur. L'environnement auteur a pour rôle d'assister l'auteur dans cette édition.

Il m'a donc semblé judicieux de séparer au cours du processus d'édition :

- Les besoins couverts par le langage de sauvegarde utilisé pour décrire les documents multimédias, et plus précisément aux formalismes d'édition (absolu, relationnel) utilisés pour spécifier et sauvegarder ces documents. Nous verrons que certains langages peuvent être une combinaison de plusieurs formalismes.
- Les besoins couverts par les environnements. Plus précisément nous identifierons les services que doivent fournir de tels environnements pour offrir à la fois une édition simple et intuitive à l'utilisateur novice et une édition puissante à l'utilisateur expérimenté.

2.1 Besoins de l'auteur couverts par les langages

Pour analyser les capacités des langages, nous allons définir un ensemble de propriétés que nous avons regroupées en deux grandes catégories :

- Expressivité, c'est-à-dire ce que le langage permet de définir comme comportements ou comme propriétés sur les objets de base d'un document.
- Capacité de spécification, c'est-à-dire la simplicité avec laquelle l'auteur peut définir ces comportements à l'aide du langage.

2.1.1 Expressivité

Comparer formellement l'expressivité des langages multimédias nécessiterait d'être capable d'identifier des classes de documents (par exemple sous forme de classes d'automates) et de définir ensuite comment chaque langage permet de couvrir partiellement ou entièrement chaque classe. Sans vouloir effectuer cet important travail théorique qui sort des objectifs de cette thèse, nous allons présenter deux types de besoins nous permettant ensuite d'identifier trois classes de documents :

Spécification des médias, qui recouvre les besoins de l'auteur en termes de définition des composants élémentaires de son document.

- *Paramétrage des attributs associés aux médias*: il est important que l'auteur puisse paramétrer les informations de visualisation de ces médias que ce soit pour les attributs spatiaux (X, Y, largeur, hauteur, couleur), mais aussi temporels (début, fin, durée).
- *Médias continus / discrets :* l'auteur a besoin de spécifier dans son document à la fois des médias discrets (images, textes) et des médias continus (sons, vidéos).
- *Médias déterministes/indéterministes :*Lorsque l'auteur inclut un média continu dans un document, il doit pouvoir spécifier le comportement de ce dernier lors de l'exécution. En effet, l'exécution de média continu ne peut être prévue statiquement si l'on désire préserver tout le contenu sémantique du média. L'auteur doit donc pouvoir spécifier le comportement qu'il désire. Deux cas sont possibles :
  - Déterministe : le système de présentation contrôle l'exécution et pour satisfaire la spécification de l'auteur il peut décider d'accélérer ou de ralentir la vitesse de présentation du média.
  - Indéterministe : le système de présentation n'aura pas de contrôle sur la présentation du média. Cela soit parce qu'au moment de jouer le média il n'y a pas de moyen de connaître la durée du média (concert en direct par exemple), soit parce que l'auteur veut délivrer complètement le contenu sémantique du média.
- *Plugins / Applets* : l'auteur peut importer dans son document des modules externes qui lui permettent d'afficher, de manipuler des données structurées ayant déjà leur propre comportement comme des scènes spécifiées en VRML[VRML99].

Les comportements, qui permettent à l'auteur de modéliser l'enchaînement temporel et spatial du document ainsi que les actions autorisées au lecteur :

- *Comportement local,* permet de définir l'enchaînement temporel et spatial entre les médias ou les groupes de médias.
- *Interactions locales*, définies lors de la spécification du document, ces interactions peuvent permettre au lecteur de déclencher des actions sur le document comme le lancement ou l'arrêt d'un objet. Ces interactions modifient le cours de l'exécution en incluant (ou en supprimant) des médias dans le cours de la présentation.
- *Navigation,* définie lors de la spécification, elle permet au lecteur de parcourir le document. Cette navigation peut être structurelle, spatiale ou temporelle. Les actions de navigation sont orthogonales au comportement temporel du document en ce sens que l'on peut définir son comportement sans tenir compte de ces interactions de navigation qui, lorsqu'elles surviendront, changeront uniquement l'instant courant de présentation du document.
- *Acquisition d'information*, ce sont tous les moyens qui permettent de récupérer des informations de la part du lecteur (champ de saisie texte, bouton de sélections multiples,).

Ces différentes classes de documents permettent à l'auteur d'écrire des documents multimédias très variés. On peut cependant les classer en trois grandes classes selon le type de comportements qu'elles permettent :

- *Les documents entièrement prédictifs* **:**dès le début de la présentation du document, on sait exactement ce qui va se produire et il n'y a pas d'interaction utilisateur. Le système de présentation peut calculer les instants de début et de fin des objets statiquement. Dans le cas particulier où ces documents contiennent des objets indéterministes, ces objets seront coupés ou maintenus à l'écran en fonction de la durée calculée initialement. Ces documents recouvrent les présentations de type film ou le lecteur n'interagit pas avec le document qui lui est présenté.
- *Les documents prédictifs avec navigation :*ces documents sont constitués d'un ensemble de sous-documents prédictifs dans lequel le lecteur peut naviguer. Cette navigation peut être de deux ordres :
  - *Navigation temporelle* **:** où le lecteur peut aller à un instant précis de la présentation.
  - *Navigation hypertexte* **:** où le lecteur passe d'un sous-document à un autre.

Ces documents recouvrent les présentations de type diaporama ou présentation hypertexte classique. La navigation ne remet pas en cause le comportement des documents.

- *Les documents interactifs*: un document interactif est un document qui implique le lecteur. Dans ce type de documents il n'est plus possible de prédire l'exécution du document car celle-ci est gouvernée par des interactions locales ou par des médias. C'est le cas par exemple d'un document très simple (voir Figure II-2) contenant une bande son en boucle infinie. Lorsque le lecteur appuie sur une touche, cela déclenche une vidéo. La fin de la vidéo interrompt la bande son, et termine le document. Les interactions du lecteur peuvent modifier localement le document et remettre en cause les durées calculées initialement. De plus, les objets indéterministes ne sont plus forcément interrompus (du fait que l'on ne soit plus contraint de connaître statiquement la durée d'un objet), on peut les laisser s'exécuter jusqu'à la fin. Le système devra être en mesure de réévaluer la suite du document pour garder les synchronisations entre les objets. Cela impose une certaine réactivité du système de présentation.

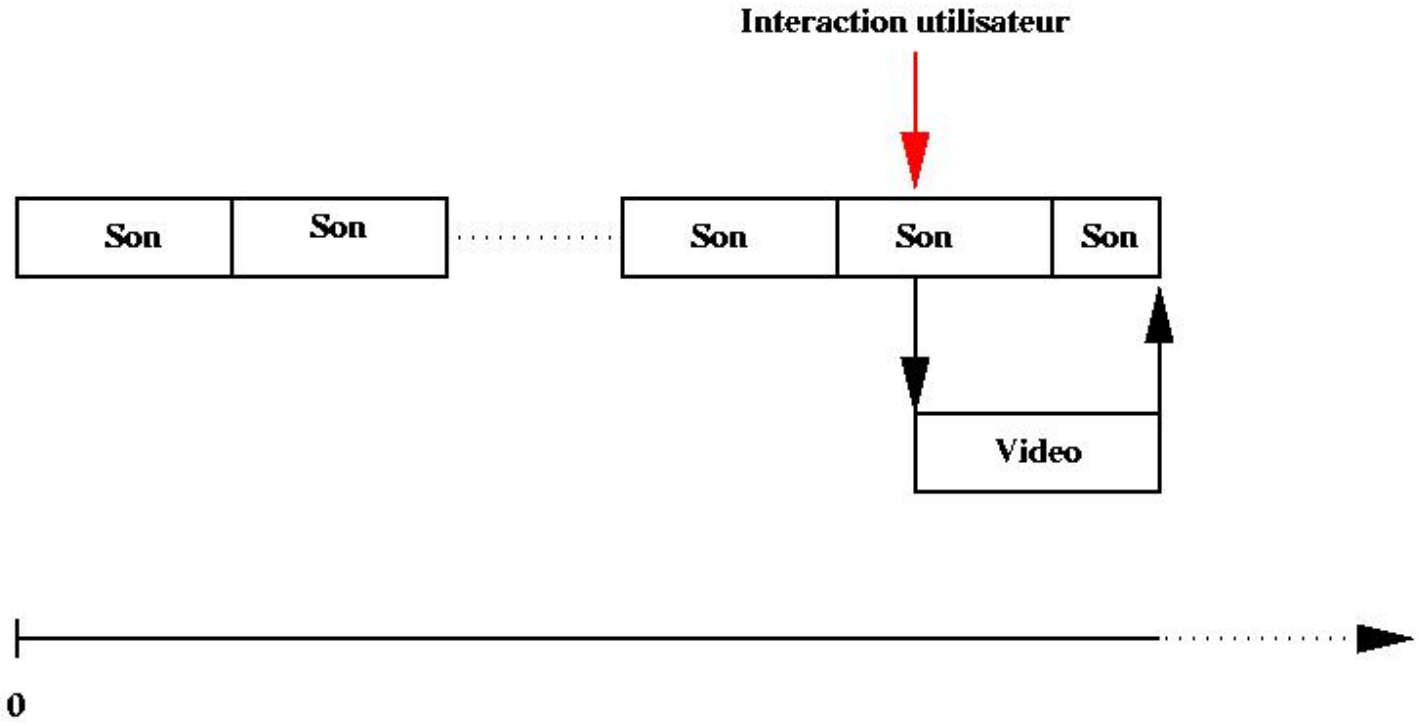


Figure II-2 : Exemple de scénario d'un document indéterministe

Nous verrons par la suite que la difficulté d'édition d'un document est liée en grande partie à la classe à laquelle il appartient.

2.1.2 Capacité de spécification

Les besoins de l'auteur lors de la spécification de documents multimédias peuvent être classés en quatre catégories :

- *Possibilité de structuration du document*pour aider l'auteur dans le découpage de son document, ceci afin de simplifier la phase d'édition. La structuration permet de définir des groupes dans un document et d'associer des propriétés communes aux objets du groupe. Elle permet ainsi de découper la tâche d'édition en sous-tâches plus simples. De plus l'auteur peut réutiliser certaines parties dans d'autres documents (ou dans le même). On peut rapprocher cette notion de celle de module dans les langages de programmation avec toutes les difficultés qui s'y rattachent (partage, mise à jour de version). Plusieurs structures peuvent cohabiter dans le document, ces structures peuvent être spatiale, hypertexte, logique mais aussi temporelle.
- *Support pour la modification du document*, pour que l'auteur puisse changer facilement la présentation

ou le contenu d'un document existant. Modifier un document consiste soit à modifier les attributs d'un objet, soit à modifier la structure du document, soit à modifier le contenu des médias. Dans tous les cas, ces modifications doivent pouvoir se réaliser sans remettre totalement en cause le document et le travail effectué jusque-là.

- *Rapidité de spécification de documents*. Lors de la phase d'édition, l'auteur doit pouvoir exprimer le comportement spatial et temporel de son document avec une difficulté proportionnelle à la complexité du comportement voulu. Il doit pouvoir par exemple exprimer des comportements complexes sans avoir à exprimer tous les comportements élémentaires nécessaires à sa réalisation. Par exemple, si l'auteur veut exprimer le déplacement d'un objet d'un point A vers un point B, il ne doit pas avoir à spécifier toutes les positions intermédiaires.
- *Support pour la spécification de documents adaptables*. L'auteur pourra définir un ensemble de comportements dépendant des conditions de présentation pour son document.

Nous allons maintenant détailler les deux derniers points.

Rapidité de spécification : c'est le besoin élémentaire qui permet, par exemple, à l'utilisateur novice de spécifier facilement un petit document en n'utilisant qu'une partie de l'environnement auteur. Sans faire un inventaire de toutes les instructions qu'aurait un langage de programmation, nous allons présenter les principales instructions utiles pour un auteur de documents :

- *Relations entre les objets* : elles permettent à l'auteur de mettre des relations prédéfinies entre les objets, lui évitant ainsi d'avoir à spécifier l'ensemble des comportements élémentaires qui sont communs à tous les documents. C'est par exemple le cas des relations temporelles *avant* et *après* ou des relations spatiales *centré* ou *aligné*. On peut noter que certaines relations sont flexibles (ou partielles), c'est-à-dire que l'auteur ne spécifie pas explicitement un unique placement temporel (ou spatial) mais un ensemble de placements possibles. Par exemple, lorsque l'auteur dit "A *avant* B", il n'explicite pas exactement le placement temporel de A, par rapport à B.
- *Evénements*: ils permettent à l'auteur de spécifier des comportements relativement complexes, en lui permettant d'associer des actions à un événement dynamique (condition externe), et dont l'instant d'occurrence n'est pas connu au moment où l'auteur écrit son document. Nous appellerons par la suite *événement*, l'événement et l'action qui lui sont associés. Nous présenterons de manière plus précise cette notion dans la section 3.3.
- *Instructions conditionnelles* : il est important que l'auteur puisse réaliser des tests, soit sur l'état de certains objets (pause, actif, affiché), sur la valeur d'attributs, sur la valeur de variables ou sur l'état du système (que ce soit le système de présentation ou le système de la machine lui-même).
- *Boucles finies ou infinies* : il peut être utile de définir des boucles pour la présentation d'un média ou d'un groupe de médias. Par exemple pour spécifier que pendant toute la présentation du document une musique de fond est jouée en boucle.

Support pour la spécification de documents adaptables :cette catégorie de besoins est celle qui consiste à écrire des documents qui s'auto-adaptent au contexte de présentation. Par contexte de présentation, nous entendons les conditions liées au système ou à l'utilisateur. Ces différents paramètres sont [Layaida99] :

- *Le système :*
  - *Le périphérique de restitution* : si l'auteur veut que son document soit affiché de façon optimale en fonction, par exemple, la taille de l'écran utilisé ou alors de façon plus générale quel que soit le périphérique de sortie (ordinateur, borne interactive,).
  - *Le débit réseau* : en cours de présentation le débit du réseau peut évoluer, l'auteur doit pouvoir exprimer comment son document réagit à ce genre d'évolution. Par exemple, si les conditions deviennent insuffisantes, il doit pourvoir changer les médias utilisés ou même changer l'agencement temporel des objets de manière à diminuer les ressources nécessaires.
  - *La charge de la machine*: de la même manière que le réseau, la charge de la machine sur laquelle est présenté le document peut évoluer. La mémoire disponible ou la charge du processeur peuvent changer en cours de présentation.

- *L'utilisateur :*
  - *La langue* : par exemple pour changer la langue de présentation de son document. Lorsque l'auteur veut faire un document dans plusieurs langues avec les mêmes enchaînements temporels et spatiaux, il peut être utile de lui permettre de spécifier un média pour chacune des langues qui l'intéressent.
  - *Le niveau d'expertise* : si l'auteur veut définir un comportement différent de son document en fonction du niveau de l'utilisateur. Par exemple on peut imaginer qu'un professeur ayant des élèves de niveaux différents oblige les élèves débutants à effectuer les exercices liés à un chapitre alors qu'il obligerait la lecture du résumé du cours pour les élèves en période de révisions.
  - *Les déficiences*: le lecteur du document peut avoir des déficiences momentanées ou permanentes. Ces déficiences visuelles ou auditives par exemple peuvent nécessiter une adaptation du document pour une lecture optimale. On peut, par exemple, imaginer que le système utilise des polices de taille plus grosse pour les personnes malvoyantes, ou qu'il remplace les médias textuels par des médias sonores.

2.2 Besoins de l'auteur couverts par l'environnement d'édition

2.2.1 Le processus d'édition

L'auteur suit le même processus d'édition quel que soit le formalisme d'édition qui lui est offert. Nous allons donc dans un premier temps nous intéresser à la définition du cycle d'édition avant de définir, dans un deuxième temps, les besoins qui en découlent.

On peut définir le processus d'édition en cinq étapes (voir Figure II-3):

1. Ouverture du document, c'est la phase pendant laquelle l'auteur ouvre un document existant (transition 1), le système vérifie sa cohérence et le formate (transition 2) c'est-à-dire qu'il calcule les placements spatiaux et temporels de chaque objet pour l'afficher à l'écran (transition 3).
2. Perception du document et navigation : c'est la phase ou l'auteur visualise chacune des dimensions (spatiale, temporelle, logique, hypertexte) du document. La présentation de ces différentes informations et les mécanismes de navigation offerts à l'auteur à l'intérieur de ces informations sont donc essentiels.
3. Modification du document, parmi ces opérations on peut distinguer :

- *Les opérations qui modifient la structure et le contenu du document*, ce sont toutes les opérations qui permettent d'ajouter ou de supprimer de nouveaux médias dans le document, ou qui permettent de structurer le document.
- *Les opérations qui modifient la présentation du document*, c'est-à-dire toutes les opérations qui modifient le placement temporel ou spatial, ou les attributs des médias. Le système applique alors les modifications sur le document et fait appel aux services de vérification de cohérence.

4. Vérification de la cohérence et formatage : cette dernière étape correspond à la réaction du système à l'opération d'édition faite par l'auteur (transition 5), c'est-à-dire que le système doit, dans un premier temps, vérifier la cohérence de cette opération, et appliquer les différents changements dans le document de manière à prendre en compte cette modification. Il doit alors afficher le résultat et le processus d'édition recommence à l'étape 2 (transition 3). C'est pour cela que l'on parle de processus d'édition cyclique.
5. Sauvegarde du document. L'environnement sauvegarde la représentation interne du document dans un fichier de sauvegarde (transitions 6 et 7).

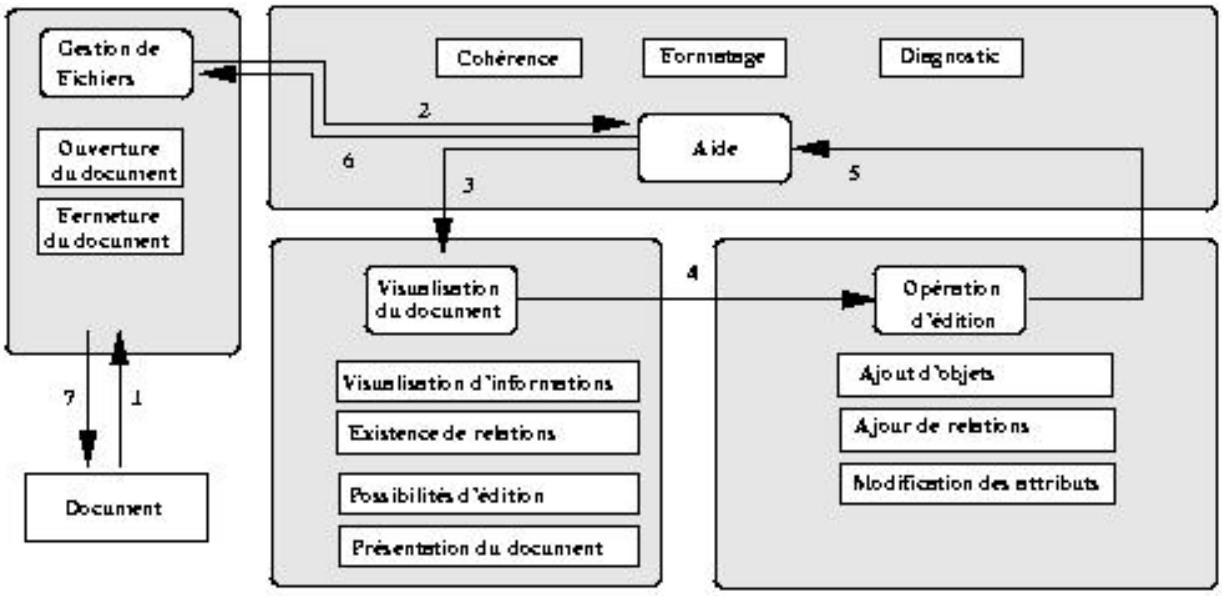


Figure II-3 : Définition du cycle d'édition d'un document multimédia

Du cycle d'édition défini précédemment, on dégage les quatre grandes fonctionnalités suivantes (voir Figure II-3):

- La gestion de fichiers (chargement, sauvegarde) ;
- La visualisation des informations contenues dans le document et la présentation du document ;
- L'édition proprement dite ;
- L'aide à l'auteur pour faciliter et automatiser certaines tâches : cohérence, formatage,

2.2.2 Les besoins des auteurs suscités par le processus d'édition

Du processus d'édition on peut dégager cinq besoins qui sont liés :

- *A trois des grandes fonctionnalités du système auteur* (visualisation, édition, aide).
- *A l'enchaînement des phases d'édition.*
- *A la vie du document en dehors d'une session d'édition* : version, stockage, diffusion, droits,

Visualisation/présentation :

- *Perception du document dans toute sa complexité :* il est important, pendant le processus d'édition, que l'auteur puisse percevoir les informations relatives aux différentes entités constituant son document. Par exemple, il doit pouvoir en fonction de ses besoins d'édition visualiser les informations spatiales et temporelles de son document. Cette visualisation ne doit pas seulement être une visualisation sous une forme textuelle ou via une palette d'attributs. Elle doit se faire selon différents modes de représentation. Ces modes doivent permettre à l'auteur de se faire une représentation mentale du document tel qu'il sera présenté au lecteur. De plus, ces représentations doivent fournir à l'auteur une explication sur le placement des objets en visualisant par exemple les relations (ou les attributs) qui induisent ce placement. Par exemple la dimension spatiale doit être visualisée telle que le lecteur la verra lors de la présentation, cette vue vérifie partiellement le principe du WYSIWYG (What You See Is What You Get) du fait des nombreux périphériques de lecture possibles pour le document. Dans le cas de la dimension temporelle, celle-ci peut être visualisée à l'aide d'une représentation graphique dans laquelle on peut référencer un temps absolu.
- *Présentation* **:**il est important que l'auteur puisse, à chaque étape du processus de conception, visualiser une des présentations possibles du document. Cette étape est essentielle à la compréhension du document car contrairement aux documents classiques, la phase d'édition ne s'effectue pas complètement sur la forme finale du document. En effet, en cours d'édition, l'auteur ne peut visionner

complètement l'aspect dynamique de la spécification de son document, il doit donc visualiser son exécution pour en avoir un aperçu complet.

- *Perception de l'espace de solutions :* l'auteur de documents multimédias de par la flexibilité des relations et la spécification partielle des médias ne définit pas une présentation, mais un ensemble de présentations possibles. Le système auteur doit rendre accessible à l'auteur cet espace de solutions.

Fonction d'édition : Le système auteur doit permettre à l'auteur d'éditer le document en fonction de ses capacités. Il doit aussi faciliter autant que possible cette tâche. De plus, il doit permettre à l'auteur d'exprimer tout ce que le langage de restitution permet de spécifier. Ces fonctions peuvent être des fonctions élémentaires comme l'insertion d'un objet dans le document, ou des fonctions plus évoluées comme la copie d'attributs de présentation ou de synchronisation. Par exemple, on peut imaginer qu'un auteur définisse un ensemble d'attributs spatiaux et temporels pour trois objets de son document, et qu'il désire appliquer le même placement spatial et temporel à trois autres objets de son document.

Aide à l'auteur:

- *Formatage*: un des intérêts d'une spécification relative du comportement des objets, est que le système peut déduire, à partir des informations données par l'auteur, le placement absolu des objets. Ce calcul, appelé *formatage,* s'applique aussi bien dans la dimension spatiale que temporelle. L'auteur n'a pas ainsi, à chaque modification de son document, à recalculer le placement de chacun de ses objets, c'est le système qui réalise cette tâche fastidieuse. Cette fonctionnalité est donc essentielle pour la visualisation des informations temporelles et spatiales du document.
- *Vérification de la cohérence* **:** lors d'une action d'édition, le système doit s'assurer que l'action n'engendre pas d'incohérence dans le document. Par exemple, s'il a été spécifié des informations contradictoires ou une référence à un objet non défini.
- *Diagnostic*: En plus de la vérification de la cohérence, le système doit aider l'auteur à comprendre les raisons des erreurs de spécification. C'est ce qu'on appelle le *diagnostic*.

Cycle d'édition :le processus de création d'un document multimédia est un processus cyclique (voir Figure II-3). C'est-à-dire que l'auteur ne spécifie pas complètement son document avant de le présenter, mais construit plutôt petit à petit son document en utilisant à chaque étape les services de visualisation et de vérification offerts par l'environnement. De ce fait, le système doit prendre en compte le fait que l'édition n'est pas un processus linéaire et doit aider l'auteur au mieux dans ce processus. Les deux points qui nous semblent essentiels dans cette tâche d'édition sont :

- *La rapidité de prise en compte des opérations d'édition*: une qualité importante d'une interface est sa capacité à répondre rapidement aux actions de l'utilisateur. Une non-réponse du système dans un délai raisonnable peut amener l'utilisateur à se lasser et à ne plus utiliser le système, ou il peut l'amener à répéter sa commande, ce qui peut avoir des effets indésirables. Il est donc important lors de la phase d'édition que le temps de prise en compte des opérations soit le plus court possible [Coutaz90]. Cela inclut la phase de vérification de cohérence.
- *La localité des modifications :* lors d'une opération d'édition, le système auteur doit modifier le moins possible la solution courante de manière à ne pas perturber l'utilisateur dans la phase de visualisation.

Cycle de vie du document :nous venons de voir les différents besoins de l'auteur lors de la spécification de son document. La vie du document ne s'arrête pas au moment où l'auteur a fini de l'écrire. Il est important que l'auteur puisse diffuser, échanger, modifier ou réutiliser tout ou partie de son document après sa spécification. Nous allons donc maintenant nous intéresser à ces différents besoins :

- *Diffusion :* un des besoins important est de permettre au document d'être largement diffusé que ce soit par CD-ROM, sur le Web, .
- *Echange avec d'autres personnes / outils :* un des besoins est l'échange des documents en phase d'édition par exemple avec d'autres personnes et/ou d'autres outils.
- *Réédition :* l'auteur peut vouloir rééditer un document, même s'il a déjà fait l'objet d'une diffusion. Il faut donc que le système auteur ait une pérennité temporelle, ou qu'il permette de sauver le document sous une forme *standard*.

- *Réutilisation de parties de document :* lors de la conception d'un document multimédia, on a souvent besoin de réutiliser certaines parties de documents précédents (voir structuration 2.1.2), il faut donc que le système permette à l'auteur d'isoler et de récupérer certaines sous-parties de documents existants. Cependant cette réutilisation n'est pas toujours une chose aisée du fait des synchronisations ou des liens externes sur certaines parties du module.

3. Les standards de documents multimédias

Comme nous l'avons dit en introduction, le document en cours d'édition est représenté par plusieurs entités physiques. Au cours de cette section nous allons nous intéresser aux langages de sauvegarde et nous les illustrerons au travers d'environnements auteur réels utilisant les formalismes du langage de sauvegarde comme formalisme d'édition.

Il existe aujourd'hui de nombreux langages pour décrire des documents multimédias, cependant ils peuvent tous se classer dans une ou dans une combinaison des quatre catégories suivantes :

- *L'approche absolue* : l'auteur décrit de façon explicite le positionnement des objets dans le temps et l'espace.
- *L'approche programmation*: l'auteur décrit le comportement de ses objets médias grâce à un langage de programmation de type impératif.
- *L'approche événementielle*: l'auteur décrit le comportement de ses objets médias grâce à des règles événement / conditions / actions.
- *L'approche relationnelle*: l'auteur décrit des relations (avant, en même temps, au-dessus,) entre les objets.

Au cours de cette analyse, nous allons présenter chacune de ces approches en les illustrant par un ou plusieurs langages représentatifs. Pour chaque approche, nous dégagerons d'un part la façon dont les langages de description de documents multimédias répondent aux besoins auteurs liés au langage (section 2.1) et d'autre part comment un environnement d'édition répond aux besoins définis dans la section 2.2.

Dans un premier temps, nous présenterons l'approche absolue utilisée notamment dans Director qui est un des outils les plus utilisés pour concevoir des documents multimédias. Dans un deuxième temps, nous présenterons les approches à base de langages de programmation en les illustrant au travers du langage Java. Dans un troisième temps, nous présenterons un langage de description à base d'événements (MHEG [MHEG93]) qui a été développé par le groupe de standardisation ISO. Nous présenterons ensuite deux approches relationnelles : SMIL, qui est une recommandation du W3C pour spécifier le comportement temporel des documents et Madeus qui est un langage à base de contraintes développé au sein du projet OPERA. Une des difficultés de l'analyse ci-dessous vient du fait que les langages actuels mélangent les différentes approches.

3.1 L'approche absolue

La caractéristique essentielle de cette catégorie est que l'auteur spécifie de façon absolue le placement spatial et temporel des objets. Cela se traduit par le fait que les instants de début des objets sont placés par rapport au début du document, et que la position spatiale des objets est donnée par rapport à un point de référence de l'écran (par exemple en haut à gauche).

Aucun standard n'existant dans cette approche, les exemples décrits ci-dessous sont basés sur une syntaxe ad hoc.

Dans la Figure II-4 on peut voir un exemple de document décrit de façon absolue. Ce document est composé de cinq éléments textes qui s'affichent successivement dans le temps toutes les 10 secondes. Les cinq éléments disparaissent ensemble de l'écran à la cinquantième seconde du document. Spatialement, les objets sont affichés les uns en dessous des autres sur l'axe Y, et les uns à côté des autres sur l'axe X.

Temporel :

```
Objet Titre1
       Début = 0
       Durée=50

Objet Titre2
       Début = 10
       Durée=40
Objet Titre3
       Début = 20
       Durée=30
Objet Titre4
       Début = 30
       Durée=20
Objet Titre5
       Début = 40
       Durée=10
```

Spatial :

```
Objet Titre1
       X = 120
       Y =  60
       Texte = "Institut"
Objet Titre2
       X = 180
       Y=80
       Texte = "National"
Objet Titre3
       X = 240
       Y=100
       Texte = "de Recherche"
Objet Titre4
       X = 300
       Y=120
       Texte = "en Informatique"
Objet Titre5
       X = 360
       Y=140
       Texte = "et Automatique"
```

Figure II-4 : Description absolue du placement temporel et spatial

L'avantage de cette approche est la facilité de description de documents simples. L'écriture de petites animations, que ce soit par spécification directe dans un fichier textuel ou par manipulation dans une interface graphique, est relativement simple. En effet, il est très simple d'imaginer une interface basée sur une métaphore *table de montage,* où l'auteur place les objets sur un axe de temps absolu.

Les inconvénients majeurs de cette approche sont :

- La difficulté de modification de ce que l'on a spécifié : par exemple, si on veut insérer le texte "INRIA" en haut de l'écran de sorte qu'il soit présent pendant toute l'animation décrite précédemment, il faut modifier le placement temporel et spatial de tous les objets. C'est-à-dire qu'il faut décaler le début de tous les objets de 10 secondes, et décaler tous les objets de 10 unités vers le bas.
- La difficulté de décrire des comportements complexes : l'auteur n'a pas d'instructions disponibles pour exprimer le comportement de son document ce qui limite son pouvoir d'expression.
- L'impossibilité de décrire des documents adaptables en fonction du lecteur ou du périphérique de présentation.
- Pas de synchronisation avec des objets indéterministes. L'auteur ne spécifiant pas de dépendance entre les objets, il ne peut y avoir de synchronisation temporelle avec la fin d'objets indéterministes.

Les générateurs automatiques de documents ([Real99]) produisent souvent des documents sous une forme absolue. De ce fait les documents produits sont relativement peu modifiables et ne sont pas rééditables.

Un outil représentatif de cette approche est Director, il permet à l'auteur de spécifier le placement temporel et spatial de ses objets de manière absolue. Il compense la limitation d'expressivité par l'utilisation de scripts. Nous présenterons Director dans la section 4.3.1.

3.2 Les langages de programmation

Une approche plus générale pour concevoir des documents multimédias est l'utilisation d'un langage de programmation. Il existe deux grandes catégories de langages de programmation :

- L'utilisation de langages de scripts intégrés ou non dans un environnement auteur. C'est par exemple le cas de Lingo intégré dans l'environnement Director.

- L'utilisation de langages plus généraux comme Java ou C++. Il existe depuis quelques années des bibliothèques (Java Media Framework [JMF00]) permettant de manipuler les différents types de médias existants réduisant ainsi le coût de conception de documents multimédias. De plus ces bibliothèques sont portables sur différentes plates-formes, permettant ainsi une plus grande portabilité du document créé. Dans les langages de programmation, on peut distinguer les langages compilés (C, C++, Pascal) des langages interprétés (Prolog, Java) :
  - Les langages compilés nécessitent une phase de compilation du programme pour toutes les plates-formes si l'on désire rendre les documents accessibles aux lecteurs. L'avantage de cette approche réside dans le fait que le document produit est adapté pour une présentation sur la plate-forme cible.
  - Les langages interprétés, quant à eux, peuvent être exécutés sur tous les systèmes d'exploitation qui possèdent une machine virtuelle de ce langage permettant ainsi au programmeur d'être déchargé de la gestion de la portabilité du code écrit. Parmi les langages interprétés Java offre aussi une autre possibilité intéressante : la définition *d'applets*. Une *applet* permet à un programme Java d'être intégré dans une page Web, le rendant ainsi accessible.

Les langages les plus adaptés à cette tâche, de par la diversité des plates-formes de conception et de restitution sont les langages interprétés. Nous allons présenter un peu plus complètement le langage Java qui est un des langages permettant la conception de documents multimédias.

En effet, Java est un langage à objets développé au début des années 90, il intègre complètement la manipulation des médias (JMF [JMF00]), des protocoles réseau et de la sécurité. Il fournit aussi un ensemble de bibliothèques évoluées (Java 2D[Java2D00], Java 3D[Java3D00]) pour les manipulations des médias ou d'objets en deux et trois dimensions.

Nous pouvons voir dans l'exemple de la Figure II-5 le document de présentation de l'INRIA. Nous avons écrit deux versions de ce document. Dans le premier cas (scène1_absolue), nous avons utilisé une spécification absolue, nécessitant une description plus longue et moins facilement modifiable. Dans le deuxième cas (scène1_relative), nous avons utilisé une spécification par relation, plus souple lors de l'édition. Notons que cette spécification nécessite l'écriture d'un formateur temporel et spatial pour calculer le placement réel des médias.

Dans un programme de description d'un document multimédia, l'auteur doit gérer explicitement la manipulation de l'environnement de présentation du document en plus du placement temporel et spatial de son document. Par exemple, il doit gérer la création d'une fenêtre dans laquelle le document va être joué. De plus, la facilité de réutilisation et de manipulation du document lors des opérations d'édition dépend énormément de la façon dont le programme a été conçu.

```
public class DocumentINRIA

void CreerMenu() {
...
}

void Scene1_absolue() {

// le constructeur de Media text prend 5 paramètres : le texte à afficher,
// les coordonnées spatiales de l'objet, l'instant de début de l'objet,
// et l'instant de fin.

MediaText T1 = new MediaText("Institut",120,60,0,50);

//Valeur, X, Y, Début, Fin

MediaText T2 = new MediaText("National",180,80,10,50);
MediaText T3 = new MediaText("de Recherche",240,100,20,50);
MediaText T4 = new MediaText("en Informatique",300,120,30,50);
MediaText T5 = new MediaText("et Automatique",360,140,40,50);
T1.Start(); T2.Start();T3.Start(); T4.Start();T5.Start ();
```

```
}

void Scene1_relative() {

// le constructeur de Media text prend 5 paramètres : le texte à afficher,
// les coordonnées spatiales de l'objet, l'instant de début de l'objet,
// et l'instant de fin.

MediaText T1 = new MediaText("Institut",120,60,0,50);

//Valeur, X, Y, Début, Fin

MediaText T2 = new MediaText("National");

// les autres attributs ne sont pas spécifié

MediaText T3 = new MediaText("de Recherche");
MediaText T4 = new MediaText("en Informatique");
MediaText T5 = new MediaText("et Automatique");

DécaléSpatialement(T1,T2,20);
DécaléSpatialement(T2,T3,20);
DécaléSpatialement(T3,T4,20);
DécaléSpatialement(T4,T5,20);

// Dans ce cas, le programmeur doit implémenter un formateur temporel
// et spatial pour calculer les attributs sur tous les médias.

T1.Start(); T2.Start();T3.Start(); T4.Start();T5.Start ();

}

static public void main(String argV) {

CreerFenetreGraphique();
CreerMenu();
Scene1_relative(); // ou Scene1_absolue();

}
```

Figure II-5 : Spécification du document INRIA en Java

L'avantage majeur de ce type d'approche est que l'auteur peut décrire n'importe quel document multimédia, puisqu'il a à sa disposition le pouvoir d'expression d'un langage de programmation.

Les inconvénients des langages de programmation sont :

- Absence de facilité de spécification, l'auteur doit programmer tous les comportements qu'il désire dans le document. Il n'existe pas, à l'heure actuelle, de boîte à outils avec de tels comportements (par exemple avec un ensemble d'opérateurs prédéfinis, comme la mise en séquence d'une liste d'objets).
- L'absence de visualisation globale du document et notamment de sa structure, et de sa dimension temporelle.
- L'inaccessibilité de l'approche à des nons-informaticiens. Cependant nous présenterons les travaux réalisés pour permettre un accès plus large à la programmation dans la section 4.2.

3.3 L'approche événementielle

Les approches dites événementielles permettent de définir des règles du type : événements/ conditions/ actions. La sémantique associée à ces règles est : si l'événement est déclenché et que les conditions sont vérifiées alors les actions sont effectuées. Les événements peuvent être associés à la fin, au début d'un objet, au changement de comportement (changement d'un des attributs caractérisant l'objet : couleur, position,), ou encore à une action utilisateur (clic sur un objet).

Le langage de ce type le plus connu est MHEG où la description du comportement spatial et temporel se fait de cette manière. Par exemple, la description de la dimension temporelle se fait au moyen d'objets composites, de primitives de composition (qui permettent de donner un comportement temporel simple à tous les objets d'un composite) et de liens. Les liens ne sont pas de simples liens hypertextes, mais permettent de

spécifier des événements temporels et spatiaux. Un lien est constitué d'une partie condition (*LinkCondition*) et d'une partie action (*LinkEffect*) qui correspond à une liste d'actions à effectuer sur une liste d'objets. Pour illustrer de manière plus précise cette approche, nous allons présenter MHML qui est une variante de MHEG dont la syntaxe a été mise sous forme XML. Nous ferons aussi le parallèle avec le modèle IMD [Vazirgianis98] qui permet de manipuler l'historique de lecture du document.

Les événements MHML

Le langage MHML [MHML98] est un langage événementiel qui utilise une syntaxe XML [XML99].

Un document MHML se compose de trois parties :

- Une partie *Area*, qui permet de définir des zones réactives pour les médias.
- Une partie *Body* qui permet de définir des groupes d'objets dans le document (opérateur *DIV*). Cette notion de groupe est relativement faible et n'est utilisée de manière directe que pour associer des comportements (qui sont définis dans la partie *Behavior*). Sur chaque objet élémentaire du document, l'auteur peut définir un ensemble d'attributs permettant par exemple de définir le placement spatial de l'objet, le style,
- Une partie *Behavior* qui permet de définir le comportement du document.

Ce comportement est composé d'un ensemble de comportements élémentaires. Un comportement élémentaire est défini par :

- **Un événement**, qui sera déclenché par le système ou par un objet du document. Un événement peut être associé à la position d'un objet, à son état d'activité, et enfin à la sélection par le lecteur d'un objet dans le document.
- **Une condition,** qui permet d'évaluer un certain nombre de paramètres sur le système ou sur des objets du document. Les conditions couvrent le même éventail de valeur que les événements, c'est-à-dire que l'utilisateur a la possibilité de tester la position d'un objet, s'il est sélectionné ou pas, s'il est en train d'être exécuté ou non. Il peut aussi combiner les conditions via les opérateurs *AND* et *OR*.
- **Une liste d'actions**, qui sera à réaliser si l'événement est déclenché et si les conditions d'application des actions sont vérifiées. Dans la liste des actions que l'utilisateur peut décrire, on trouve le déclenchement et l'arrêt d'un objet ainsi que le changement d'attributs spatiaux.

La partie *Behavior* comporte un événement particulier, le *Start_Link* qui permet de définir les objets qui sont lancés au début de la présentation.

Dans l'exemple de la Figure II-6 nous présentons en MHML le même exemple de document que celui de la Figure II-3.

```
<?xml version="1.0"?><!DOCTYPE MHML PUBLIC "mhml.dtd" "mhml.dtd" >

<MHML>
<BODY>

<DIV y="0" x="0" width="640" height="480" id="Document"> <TXT y="0" x="0" width="100"
height="100" id="INRIA"/> INRIA </TXT>
       <DIV y="0" x="0" width="640" height="480" id="Groupe">
              <TXT y="180" x="80" id="INRIA0"/> Institut </TXT>
              <TXT y="240" x="100" id="INRIA1"/> National </TXT>
              <TXT y="300" x="120" id="INRIA2"/> de Recherche </TXT>
              <TXT y="360" x="140" id="INRIA3"/> en Informatique</TXT>
              <TXT y="420" x="160" id="INRIA4"/> et Automatique </TXT>
       </DIV>
</DIV>
</BODY>
<BEHAVIORS>
<START_LINK>
       <ACTION>
```

```
            <RUN propagation="true" target="Document"/>
      </ACTION>
</START_LINK>

<LINK div="Document">
      <RUN_EVENT target="Document" value="true"/>
      <ACTION>
            <RUN propagation="true" target="INRIA0"/>
      </ACTION>
</LINK>

<-- Evénement exemple -->
<-- Définition d'un événement sur le composite groupe
Cet événement est un événement temporel qui sera déclenché  au début du document (time equal
O). L'action associée à cet événement sera une action de présentation (run), qui aura pour
effet de démarrer l'objet INRIA0. La valeur propagation = true, signifie que si l'objet
INRIA0 est un objet composite, cet événement sera automatiquement propagé à ses fils. -->

<LINK div="Groupe">
<TIME_EVENT target="INRIA0" operator="equal" value="0" />
      <ACTION>
            <RUN propagation="true" target="INRIA1"/>
      </ACTION>
</LINK>
<-- fin de l'événement exemple -->

<LINK div="Groupe">
      <TIME_EVENT target="INRIA0" value="10" operator="equal"/>
      <ACTION>
      <RUN propagation="true" target="INRIA2"/>
      </ACTION>
      </LINK>
<LINK div="Groupe">
      <TIME_EVENT target="INRIA0" value="20" operator="equal"/>
      <ACTION>
      <RUN propagation="true" target="INRIA3"/>
      </ACTION>
      </LINK>



<LINK div="Groupe">
      <TIME_EVENT target="INRIA0" value="30" operator="equal"/>
      <ACTION>
      <RUN propagation="true" target="INRIA4"/>
      </ACTION>
      </LINK>
<LINK div="Groupe">
      <TIME_EVENT target="INRIA0" value="40" operator="equal"/>
      <ACTION>
      <RUN propagation="true" target="INRIA5"/>
      </ACTION>
      </LINK>
<LINK div="Document">
      <TIME_EVENT target="INRIA" value="50" operator="equal"/>
      <ACTION>
      <STOP propagation="true" target="INRIA"/>
      <STOP propagation="true" target="INRIA0"/>
      <STOP propagation="true" target="INRIA1"/>
      <STOP propagation="true" target="INRIA2"/>
      <STOP propagation="true" target="INRIA3"/>
      <STOP propagation="true" target="INRIA4"/>
      </ACTION>
      </LINK>
</BEHAVIORS>
</MHML>
```

Figure II-6 : Spécification événementielle du document INRIA en MHML

On peut noter que le modèle IMD proposé par Varzigianis [Vazirgianis98] proche de MHML permet en plus de tester l'historique des événements. Un des intérêts est de pouvoir ainsi faire évoluer le document en

fonction des actions précédentes de l'utilisateur. Cela se fait essentiellement par une extension de la partie condition sur les événements. Ces conditions sont une combinaison d'expressions booléennes et d'opérateurs sur les conditions de déclenchement des événements. Les opérateurs sont :

- Any (k,e_1, .., e_n) : vrai si au moins k événements parmi les n événements e_1, .., e_n ont été déclenchés.
- Seq(e_1, .., e_n) : vrai si les événements ont été déclenchés en séquence.
- Times (n, e_1) : vrai si l'événement e_1 est arrivé n fois.
- In(e_1 ,Int) : vrai si l'événement e_1 est intervenu dans l'intervalle Int.
- Not(e_1 ,Int) : vrai si l'événement e_1 n'est pas intervenu dans l'intervalle Int.
- S_CON(e_1, .., e_n) : vrai si les événements e_1, .., e_n sont arrivés en séquence.

Synthèse sur l'approche événementielle

Les langages événementiels ont un pouvoir d'expression très important et permettent de décrire tous les types de documents.

La difficulté majeure de cette approche réside dans la maîtrise du comportement du document. En effet, avoir une vue globale du comportement est relativement difficile à cause de l'aspect imprédictif de certains événements et de par le grand nombre d'événements qu'il faut spécifier pour décrire le comportement de ce document.

Aujourd'hui peu de systèmes auteur permettent d'avoir une édition évoluée de tels langages, on se ramène souvent à de la programmation classique (voir Lingo de Director). Les facilités de spécifications de l'auteur sont relativement faibles. Nous verrons dans le Chapitre VI, avec la présentation de MHML-Editor, ce qu'il est possible de faire pour aider l'auteur avec ce type de langage en lui offrant notamment, un aperçu des différents comportements possibles de son document.

3.4 L'approche relationnelle

Les approches relationnelles visent à faciliter l'écriture des documents multimédias en permettant à l'auteur de spécifier des relations entre les objets, par exemple :

- Enchaînement temporel de type séquence : les objets sont joués les uns après les autres.
- Position spatiale définie par des placements relatifs de type centrage, alignement,

L'auteur ne spécifie donc pas de manière absolue la position spatiale et temporelle de tous les objets, c'est le système qui, à partir de toutes les informations qu'il possède, calcule ce positionnement (étape de formatage). Les relations introduites peuvent être flexibles ou non.

Ce type de relation permet de définir un ensemble de placements possibles, les relations non flexibles permettent elles de définir des placements relatifs fixes. Par exemple, la relation *during* permet de spécifier qu'un objet A sera pendant un objet B, sans pour autant spécifier explicitement où se situent ces deux objets l'un par rapport à l'autre.

Le fait d'introduire des relations flexibles permet de modéliser des documents plus facilement adaptables.

Ces différents langages illustrant cette approche (CMIFed [Hardman93], ISIS [Kim95], TIEMPO [Wirag95]) permettent plus facilement à l'auteur de modifier le document : en effet, lors de l'ajout ou du retrait d'un objet, c'est le système qui s'occupe de calculer le nouveau placement de tous les objets. On peut noter dans ces approches deux catégories : la première basée sur des arbres d'opérateurs (SMIL, CMIF), permet à l'utilisateur de définir par exemple un arbre temporel où les noeuds représentent des relations et les feuilles des médias. Nous illustrerons cette approche par la présentation de SMIL (section 3.4.1). La deuxième catégorie d'approche, permet à l'auteur de définir des groupes d'objets ainsi que de définir des relations binaires entre les objets d'un même groupe (ISIS, TIEMPO, MADEUS [Layaïda97]). Nous présenterons Madeus (section 3.4.2) pour illustrer cette approche.

3.4.1. Les arbres d'opérateurs : SMIL

Nous allons illustrer cette approche à l'aide du langage SMIL [SMIL98].

SMIL est une recommandation du World Wide Web Consortium (W3C), sa version courante est la 1.0. Sa syntaxe est décrite par une DTD XML. La version 2.0 de SMIL est prévue pour février 2001 [SMIL2-00].

L'objectif de SMIL 1.0 est de faciliter l'intégration de médias de différents types en fournissant un moyen de spécifier à la fois les dimensions spatiale et temporelle du document.

Nous allons rapidement en présenter les principales caractéristiques : un fichier SMIL 1.0 (Figure II-7) est composé d'un élément *head* et d'un élément *body*. La partie *head*, qui correspond à l'en-tête du fichier permet de spécifier les informations non temporelles du document. Par exemple, le positionnement spatial des objets se fait au moyen de régions. Une *région* définit une zone d'affichage à l'écran. Cette zone d'écran peut être partagée par plusieurs objets multimédias. Dans l'exemple de la Figure II-7, la *région* "droite" est partagée par deux objets (une image et un texte) décrits dans la partie *body*.

La partie *body*, qui correspond au corps du document, contient la description de l'ensemble des objets manipulés dans le document, la description de leur organisation temporelle et la définition des liens hypertexte contenus dans le document.

```
<?xml version="1.0"?>

<smil>
<head>
<layout>
       <region id="région1" top="180" left="80" width="100" height="40" />
       <region id="région2" top="240" left="100" width="100" height="40" />
       <region id="région3" top="300" left="120" width="100" height="40" />
       <region id="région4" top="360" left="140" width="100" height="40" />
       <region id="région5" top="420" left="160" width="100" height="40" />
</layout>
</head>

<body>
<switch id="Contentenu adaptatif">
       <par dur="50" id="Document" system-bitrate="115200">
              <text src="/home/Texte1.html" id="T1" region="region1"/>
              <text src="/home/Texte2.html" id="T2" begin="10.0"
              region="region2"/>
              <text src="/home/Texte3.html" id="T3" begin="20.0"
              region="region3"/>
              <text src="/home/Texte4.html" id="T4" begin="30.0"
              region="region4"/>
              <text src="/home/Texte5.html" id="T5" begin="40.0"
              region="region5"/>
       </par>
       <par dur="50" id="Document" system-bitrate="144000">
              <sound src="/home/MusicINRIA.mp3" >
              <text src="/home/Texte1.html" id="T1" region="region1"/>
              <text src="/home/Texte2.html" id="T2" begin="10.0"
              region="region2"/>
              <text src="/home/Texte3.html" id="T3" begin="20.0"
              region="region3"/>
              <text src="/home/Texte4.html" id="T4" begin="30.0"
              region="region4"/>
              <text src="/home/Texte5.html" id="T5" begin="40.0"
              region="region5"/>
       </par>
</switch>
</body>
</smil>
```

Figure II-7 : Spécification relationnelle du document INRIA en SMIL

L'organisation de la partie *body* des documents SMIL est basée sur une structure hiérarchique dont les noeuds sont des opérateurs temporels et les feuilles des médias. L'opérateur temporel associé à un noeud décrit la façon dont s'enchaînent temporellement les fils du noeud. Les opérateurs peuvent être soit l'opérateur *par*, soit l'opérateur *seq*. L'opérateur *par* permet de décrire une exécution en parallèle des fils, l'opérateur *seq* permet de décrire une exécution en séquence des fils. Dans l'exemple de la Figure II-7 on peut voir un exemple d'utilisation de ces deux opérateurs.

L'auteur peut compléter/modifier l'information déduite des opérateurs en spécifiant des attributs sur les médias. Par exemple, l'attribut *dur* permet de spécifier explicitement la durée d'un objet, alors que par défaut les objets discrets ont une durée infinie. Les attributs *begin* et *end* permettent, eux, de spécifier de façon absolue ou relative (par rapport au début ou à la fin d'autres objets dans le document) le début ou la fin des objets. Le placement de ces attributs permet de définir des synchronisations temporelles plus fines entre les objets.

Exemple a :

```
<par>
       <img src="" id="A" dur="6s">
       <img src="" id="B" dur="4s">
       <img src="" id="C" dur="3s">
</par>
```

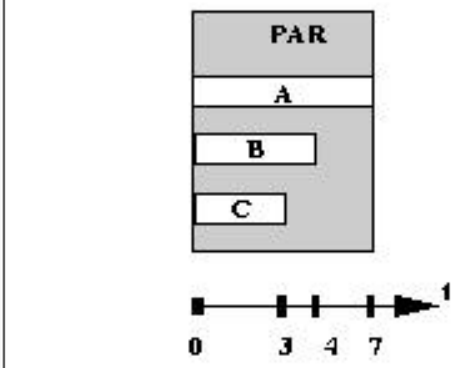


Figure II-8 : L'attribut dur de SMIL

Dans l'exemple de la Figure II-8, on peut voir un exemple de noeud *par* avec trois fils. Sur chacun de ses fils, l'attribut *dur* est spécifié de façon explicite, comme il n'y a pas d'attribut *begin* de spécifié, les trois objets commenceront en même temps. Cela signifie qu'on laisse aux trois objets le placement défini par défaut par le noeud *par*. Les trois objets vont se jouer en parallèle, ils commenceront en même temps, et ils se termineront quand leur durée respective sera écoulée.

Exemple b :

```
<par>
       <img src="" id="A" dur="6s" begin="3s">
       <img src="" id="B" dur="4s">
       <img src="" id="C" dur="3s"
       begin="end(B)">
</par>
```

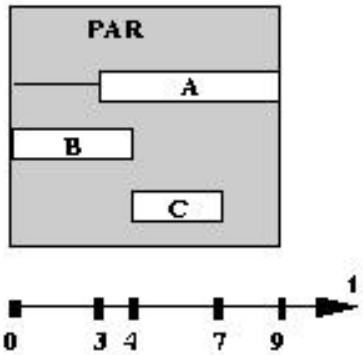


Figure II-9 : Les attributs begin et end de SMIL

Dans l'exemple de la Figure II-9, on peut voir trois manières de spécifier le début d'un objet :

- de façon absolue, il permet de retarder le début effectif de l'objet ;
- de façon implicite, dans ce cas, l'objet démarre en même temps que son ascendant (si c'est un noeud *par*) ;

- de façon relative, en liant le début de l'objet au début ou à la fin d'un autre objet.

Le comportement par défaut a été complètement modifié par la définition de l'attribut *begin* sur les objets A et C. Le début de l'objet A a été retardé de trois secondes par rapport au début du noeud *par* et l'objet C commence à la fin de l'objet B.

L'attribut *endsync* sur l'élément *par*, permet de lier la fin de l'élément composite à la fin d'un des objets qui le compose. Par exemple, la valeur *first* pour l'attribut *endsync* signifie que l'objet *par* se termine avec la fin du premier objet qui le compose. Dans le cas où les fils sont des objets déterministes cette valeur est connue statiquement, dans le cas d'objets indéterministes cette valeur ne sera connue que dynamiquement.

En résumé, la structure d'opérateurs permet à l'auteur de spécifier des comportements temporels basiques. Cependant, pour décrire des comportements temporels plus complexes, l'auteur doit manipuler les attributs *begin* et *end* sur les objets.

On peut noter que le langage SMIL offre un attribut pour permettre un certain type d'adaptation. L'attribut *Switch*, permet de définir une alternative en fonction des conditions de présentation du lecteur. Par exemple, cela permet de définir deux branches au document, une si l'utilisateur utilise un modem, l'autre s'il est sur un réseau local par exemple. Dans chacune des branches, on peut utiliser des médias différents et adaptés au contexte. Cet opérateur permet de définir une adaptation sommaire qui revient en partie à décrire statiquement les différents comportements possibles. On peut voir dans l'exemple de la Figure II-7 l'utilisation de cet élément. Dans le cas où le lecteur a un débit réseau supérieur à 144000 baud/s nous ajoutons un élément audio à la présentation.

Dans l'exemple de la Figure II-7, nous avons défini en SMIL le document INRIA en privilégiant l'utilisation de valeurs d'attribut relatives (nous avons défini que tous les objets finissaient en même temps par le positionnement de l'attribut *dur* sur le noeud Document). En effet, nous aurions pu construire le même document en n'utilisant que des valeurs absolues (positionnement des attributs *dur* sur tous les médias). L'avantage d'une spécification plus relative est qu'elle est plus facilement modifiable. Ainsi si nous voulons insérer un nouvel objet "INRIA", présent tout au long de la scène, nous avons juste à le spécifier (avec l'instant de début désiré) dans l'objet *par*. Nous n'avons alors pas à modifier la durée des objets, celle-ci étant définie de façon relative.

On peut noter aussi que l'approche à base d'arbres d'opérateurs n'est pas vraiment une approche relationnelle pure, puisqu'un objet ne peut être impliqué que dans une seule relation. De ce fait, lorsque l'on désire ajouter une relation, on est souvent amené à restructurer l'arbre d'opérateurs.

La difficulté majeure avec ce genre d'approche est de bien concevoir son document pour permettre des modifications futures. Cependant, d'une part, l'auteur ne peut prévoir a priori toutes les modifications futures qu'il voudra apporter à son document, d'autre part il arrive parfois qu'un bon document pour une modification ne le soit pas pour une autre modification. Pour toutes ces raisons les remises en cause de la structure de l'arbre d'un document SMIL sont fréquentes, et nous savons que restructurer un arbre n'est pas une chose simple [Quint87].

Cependant, un des avantages majeurs de cette approche est qu'elle permet de définir rapidement un comportement temporel simple.

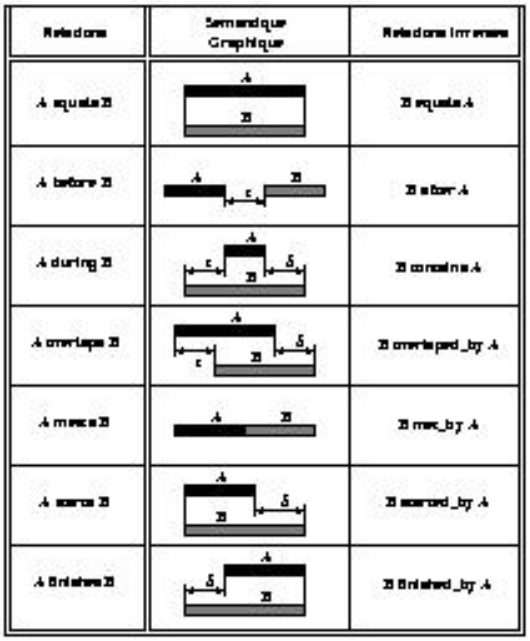
On peut noter pour finir qu'un langage comme SMIL permet de définir des documents intégrant des médias indéterministes tout en gardant certaines synchronisations (attribut *endsync*).

3.4.2 Une approche à base de relations binaires : Madeus

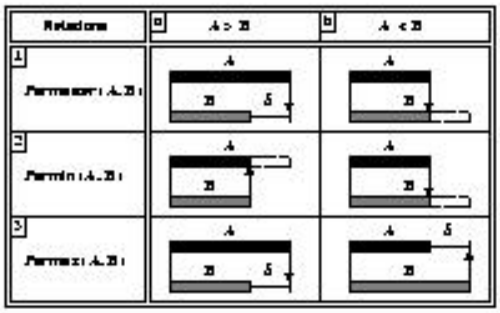
Le langage Madeus [Layaïda97] a été développé au sein du projet Opéra. Les documents Madeus sont composés de 4 sections :

- **La section média** : permet de définir les médias qui seront utilisés dans le document. Ces médias seront référencés dans les sections spatiales et temporelles.

- **La section spatiale** : permet de définir les attributs de présentation graphique de l'objet (fonte, couleur, placement, etc.). On peut aussi associer à chaque objet ou groupe d'objets des relations spatiales pour définir un placement relatif entre chaque élément.
- **La section temporelle** : permet de définir la position temporelle de tous les objets du document en plaçant des relations temporelles entre les objets ou entre groupes d'objets. Les relations sont basées sur les relations d'Allen quantifiées (Figure II-10a) augmentées de trois relations supplémentaires exprimant des interruptions (Figure II-10b).
- **La section hypertexte** : permet de définir des liens entre les objets ; ils peuvent être entre documents ou temporels. Cette section est incluse dans la partie temporelle. L'auteur peut aussi définir des liens temporels. Un lien temporel permet d'aller vers un instant donné de la présentation.



a) Les relations d'Allen



b) Les relations causales

Figure II-10 : Les relations temporelles de Madeus

Si nous reprenons l'exemple de notre petite animation présentant l'INRIA (Figure II-11) dans le langage Madeus, on peut voir les trois parties du document : la description des médias, la description de la structure temporelle et spatiale. Dans ces deux structures, le positionnement des objets est défini par des relations entre les objets (comme *Finishes*, *AlignéAGauche*). La principale différence avec SMIL est que l'on a des groupes d'objets et non pas un arbre d'opérateurs. De ce fait si l'on désire changer le placement temporel des objets pour réaliser deux séquences en parallèle, il suffit de changer les relations entre les objets, et l'on n'a pas à changer la structure du document. Une limite du langage réside dans la localité des relations, une relation ne peut porter que sur des objets présents dans le même groupe. Cette limitation permet de conserver une localité à la spécification. L'auteur n'a ainsi pas à comprendre la structure complète de son document pour comprendre le comportement local de ses médias.

```
<Madeus>

<Media>
       <text Id= "O1" value="Institut" >
       <text Id= "O2" value="National" >
       <text Id= "O3" value=" de Recherche " >
       <text Id= "O4" value="en Informatique" >
       <text Id= "O5" value="et Automatique" >
</Media>

<Spatial>
       <Groupe Id="Animation">
              <Zone ID="z1" media="O1" X="60" Y="120">
              <Zone ID="z2" media="O2">
              <Zone ID="z3" media="O3">
              <Zone ID="z4" media="O4" >
              <Zone ID="z5" media="O5" >
       <Relation>
              <AligneAGauche param1="z1" param2="z2" delta="+60">
              <AligneAGauche param1="z2" param2="z3" delta="+60">
              <AligneAGauche param1="z3" param2="z4" delta="+60">
              <AligneAGauche param1="z4" param2="z5" delta="+60">
              <Under param1="z1" param2="z2" delta="+20">
              <Under param1="z2" param2="z3" delta="+20">
              <Under param1="z3" param2="z4" delta="+20">
              <Under param1="z4" param2="z5" delta="+20">
       </Relation>
       </Groupe>
</Spatial>

<Temporal>
       <Groupe Id="AnimationT" dur="50s">
              <Zone ID="T1" X="60" Y="120" media="O1">
              <Zone ID="T2" media="O2">
              <Zone ID="T3" media="O3">
              <Zone ID="T4" media="O4">
              <Zone ID="T5" media="O5">
       <Relation>
              <finishes param1="z1" param2="z2" delay="10s">
              <finishes param1="z2" param2="z3" delay="10s">
              <finishes param1="z3" param2="z4" delay="10s">
              <finishes param1="z4" param2="z5" delay="10s">
       </Relation>
</Groupe>
</Temporel>
</Madeus>
```

Figure II-11 : Spécification relationnelle du document INRIA en Madeus

Le langage Madeus ne permet de décrire que des documents prédictifs avec des liens de navigation. L'introduction des objets indéterministes lève de nouveaux problèmes pour le formatage ([Layaïda97]) non encore résolus de manière satisfaisante à ce jour.

Un des avantages majeurs de l'approche est qu'elle permet à l'auteur de décrire des relations flexibles. Par exemple, cela permet d'utiliser ces informations pour adapter automatiquement le document au contexte de présentation. Nous verrons au cours de la présentation du système Madeus (section 4.3.3), des travaux réalisés dans ce sens.

Un autre avantage est la facilité de modification due en partie aux intervalles de durées définis sur les objets. En effet, cela permet au système d'adapter la durée des objets au cours du processus d'édition (en fonction de l'intervalle donné par l'auteur) évitant ainsi à l'auteur de modifier la durée et le placement de ces objets lors de chaque modification de son document.

3.4.3 Synthèse sur les approches relationnelles

La principale limite dans les approches relationnelles actuelles est le pouvoir d'expression. C'est-à-dire

qu'elles ne permettent pas de spécifier des interactions complexes entre l'utilisateur et le document, ou des enchaînements complexes comme des répétitions conditionnelles. Cependant, si l'on reste dans le cadre de documents prédictifs, l'extension de ces langages à une expressivité plus grande ne pose pas de problème. Dans ce cas, les langages relationnels ont le même pouvoir d'expression que des langages comme MHML (restreint à un comportement prédictif). L'avantage par rapport à des approches événementielles est la présence de relations de haut niveau qui permettent de définir des comportements entre les objets. Cela permet d'avoir une description du comportement plus concise. De plus la sémantique de la structuration du document est plus forte. On peut noter une différence entre l'approche de SMIL où la structuration est une structuration temporelle à base d'arbre d'opérateurs, et celle de Madeus où il existe une structure temporelle indépendante des relations temporelles à base de groupes. La structure à base d'arbre d'opérateurs peut être un avantage pour la description de documents simples [Bétrancourt99], cependant la structure de groupes permet une plus grande souplesse pour les modifications et l'évolution du document.

Un autre avantage de ces approches est la définition des objets dans des intervalles de valeurs. Cela rend la spécification plus facilement adaptable, et permet d'automatiser le processus de formatage. Une proposition est faite par Kim [Song99] pour intégrer cette notion dans SMIL-2.0 après l'avoir intégrée dans MPEG 4.

L'approche relationnelle relativement récente n'a pas encore fait ses preuves, notamment pour des documents complexes. Cependant, du point de vue de l'édition, cette approche semble offrir de plus grands potentiels du fait du bon compromis qu'elle offre entre facilité de spécification (et de réédition) et pouvoir d'expression [Jourdan98c]. Parmi les outils mettant en oeuvre une telle approche, on peut citer Grins[Bulterman00] et Madeus que nous présenterons dans les sections 4.3.2 et 4.3.3. Elle a suscité de nombreux travaux de recherche ([Layaïda97], [Kim95], [Hardman93], [Wirag95]) du fait des nombreux aspects non encore résolus qu'elle soulève, comme le formatage efficace ou dynamique et la cohérence. En effet, à ce jour il n'existe pas de système auteur basé sur ce type d'approche offrant les services d'édition adaptés.

3.5 Synthèse sur les langages de documents multimédias

Nous venons de voir que l'auteur de documents multimédias dispose d'un ensemble de formalismes pour spécifier ses documents. Au cours de cette analyse, nous avons pu voir les différences qu'il existe entre ces approches en terme d'expressivité et d'utilisation :

- Les approches absolues permettent la description de documents sans synchronisation temporelle relative. Cela signifie que dans le cas de médias imprédictifs l'auteur ne peut définir de synchronisation temporelle avec d'autres médias. Elles sont limitées en expressivité, mais sont simples d'utilisation.
- Les approches programmation et événementielle sont très expressives, mais difficiles d'utilisation. Ces approches permettent de décrire les trois classes de documents décrites dans la section 2.1.1.
- Les approches relationnelles offrent le meilleur compromis entre expressivité et simplicité du fait qu'elles permettent de spécifier un large éventail de documents (prédictif avec navigation) de manière relativement simple et intuitive. Parmi ces approches, on peut distinguer deux sous-classes :
    - Les arbres d'opérateurs, qui au profit de la simplicité rendent certaines opérations d'édition plus complexes. Notons que dans SMIL 2.0 il sera intégré de manière explicite les objets indéterministes et les interactions.
    - Les relations dans des groupes, qui offrent une plus grande souplesse de modification. Dans Madeus des travaux pour intégrer l'indéterminisme sont en cours.

On peut noter que l'on peut combiner ces deux approches (voir Chapitre IV). Nous allons maintenant présenter les différents outils d'édition de documents en nous intéressant essentiellement aux formalismes proposés par ceux-ci. Ceci afin de voir comment ils répondent aux besoins définis dans la section 2.

4. Les outils d'édition

L'objectif est d'identifier les fonctions générales offertes par les environnements d'édition pour les confronter aux besoins vus en section 2.2. Cependant, cet exercice présente un certain nombre de limites du fait de la dépendance entre l'outil et le langage de sauvegarde utilisé.

Les outils d'édition de documents multimédias sont issus de plusieurs origines, les principales sont listées ci-dessous en fonction de la nature des données manipulées :

- *Les documents textuels* : comme Thot[Quint99] ou Word qui permettent d'introduire tous les types de médias dans leurs documents (sans synchronisation temporelle entre eux) ainsi que la définition de synchronisations spatiales simples entre ces médias (alignement, centrage).
- *Les documents hypertextes* : avec l'avènement d'internet, des outils accompagnent l'émergence de standards (DHTML[DHTML98], flash[Macromedia-flash00], SMIL), c'est le cas par exemple de RealNetworks [Real98] et d'outils comme SmilWizard [Real99].
- *Les bases de données* : qui se tournent de plus en plus vers des outils plus complets, outils qui permettent aujourd'hui de construire des requêtes complexes et de les visualiser sous une forme multimédia [Martin99].
- *Les documents multimédias*: depuis l'avènement de ce type de documents, des outils ont permis de les éditer et ont évolué avec les besoins des auteurs, c'est le cas par exemple de Director et de Toolbook [Asymetrix99] qui offrent des outils complets.
- *Les langages de programmation* : qui offrent de plus en plus de facilités pour manipuler les médias et l'émergence de boîtes à outils complexes, c'est par exemple le cas de Java et des JMF.

À partir de ces différentes origines nous avons choisi de classer les environnements en trois grandes catégories que nous présenterons ensuite :

- Les approches à base de schémas, où les auteurs ne font que remplir des documents prédéfinis. Cette catégorie regroupe les outils issus des documents structurés qui ont eut le souci de simplifier au maximum la tâche de l'auteur (section 4.1).
- Les approches à base de programmation, où l'auteur spécifie un document comme une application à part entière (section 4.2).
- Les approches classiques, issues de l'édition des documents textuels ou multimédias, où l'auteur définit de bout en bout son document avec l'aide du système (section 4.3).

4.1 Les environnements auteur à base de modèles prédéfinis

La première approche que nous allons présenter est celle qui utilise des schémas prédéfinis. L'idée générale de ces outils est de fournir à l'auteur des types de présentations prédéfinies, comme, par exemple, une succession d'images avec un fond sonore à partir duquel il n'a plus qu'à instancier les médias. L'intérêt de ces approches est de simplifier au maximum l'édition de documents multimédias en fournissant des modèles prédéfinis pour les principales classes de documents, en évitant ainsi à l'auteur d'avoir à manipuler un document multimédia dans toute sa complexité.

4.1.1 PowerPoint

Le premier exemple, sans doute le système le plus connu, combine une approche classique d'édition et une édition à base de schéma. Pour la création de transparents, le système offre à l'auteur un ensemble de transparents prédéfinis (Figure II-12). L'auteur n'a alors plus qu'à remplir les différents champs de son transparent. Le système permet à l'auteur de modifier la présentation de son transparent à souhait, en enlevant ou en rajoutant certains champs. Le schéma temporel entre les différents transparents reste élémentaire (séquence), cependant l'auteur peut obtenir un schéma temporel plus évolué à l'intérieur de ses transparents, en décrivant des comportements temporels simples (séquence, animation). Pour cela, il dispose d'un ensemble

d'animations prédéfinies, ainsi que de la possibilité de décrire l'ordre d'apparition des objets.

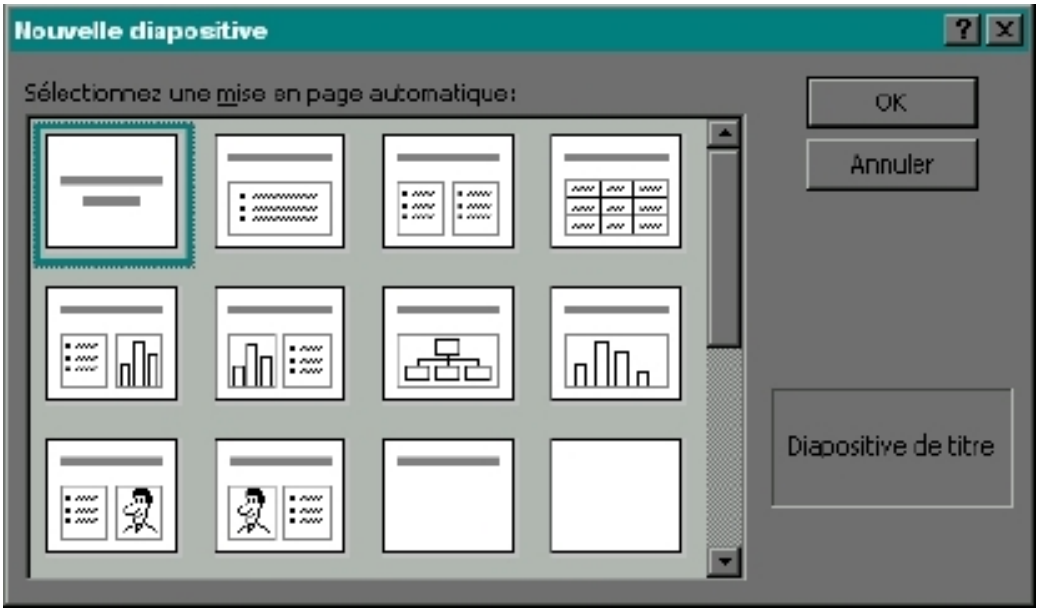


Figure II-12 : Le jeu de transparents prédéfini de PowerPoint

4.1.2 Smil Wizard

Le deuxième exemple d'outil à base de schémas prédéfinis est SMIL Wizard [Real99] (Figure II-13). L'objectif de ce système est d'offrir un moyen simple d'écrire des documents SMIL pour des personnes n'ayant aucune connaissance du langage. Le système propose un ensemble de présentations prédéfinies : Karaoké, succession de transparents, présentation de photographies,
L'auteur choisit le type de présentation (Figure II-13a) qu'il désire, et ensuite il spécifie les informations sur la localisation des médias (Figure II-13b).
Le système génère ensuite un document SMIL présentant les informations comme l'auteur l'a souhaité.
L'inconvénient majeur de ce type d'environnement est qu'il ne permet pas de définir ses propres présentations ou de sortir légèrement de la présentation standard proposée par le système d'édition. Un autre inconvénient est la qualité du fichier produit. Le système produit lors de la génération automatique un fichier utilisant peu les possibilités du langage, notamment pour la structuration de l'information (le fichier produit est constitué d'un élément *par*, et tous les objets sont placés de manière absolue grâce à des attributs *begin*), de ce fait, il est difficilement rééditable par un auteur.

Figure II-13 : Les vues d'édition de Smil Wizard

4.1.3 RealSlideShow

Le dernier environnement que nous présentons est RealSlideShow [Real98], il permet de générer des présentations sous forme de transparents. Chaque transparent peut contenir une image et du texte. L'auteur peut choisir la durée de chacun de ces transparents, et de mettre un son pendant la présentation des transparents.

On peut voir, dans la Figure II-14 les deux vues de RealSlideShow :

- la *vue temporelle* (Figure II-14a) qui permet de définir les transitions entre les transparents et la durée de chacun des transparents ;
- *la vue région* (Figure II-14b) qui permet de définir un agencement spatial pour les objets.

Comme pour le système précédent, l'auteur est guidé lors de la conception de sa présentation, mais dans ce cas-ci, il peut modifier certains paramètres de la présentation, comme l'agencement spatial des objets.

Figure II-14 : Les vues d'édition de RealSlideShow

Le langage de sortie de RealSlideShow est un langage propriétaire qui ne peut être lu que par les outils de RealNetworks. C'est à dire que les médias utilisés sont dans un format propriétaire.

4.1.4. Synthèse sur les environnements à base de modèles de présentation prédéfinis

Les environnements d'édition à base de modèles prédéfinis permettent à l'auteur d'écrire facilement des documents multimédias. Du point de vue de l'édition, le système offre très peu de possibilités à l'auteur pour spécifier un placement qui n'est pas prédéfini. Les besoins en visualisation et en aide sont moins importants que dans le cadre d'un environnement classique : l'auteur ne peut pas ici spécifier de documents incohérents du fait des restrictions et des contraintes de manipulations permises à l'auteur. L'édition devient minimale. Le système d'édition réduit volontairement le pouvoir d'expression du langage sous-jacent pour simplifier la tâche de l'auteur. Dans le cas de PowerPoint, la seule possibilité de spécification d'un comportement non défini est d'utiliser un formalisme de placement absolu. Dans le cas des deux derniers outils, les possibilités de l'auteur sont tellement réduites qu'il n'a que peu de moyen pour modifier son document une fois le processus d'édition fini. En effet, le document de sauvegarde ne contient aucune des informations qui ont permis l'édition du document.

De plus, les documents produits sont de qualité relativement pauvre, et peuvent être difficilement réutilisés dans un autre contexte. L'utilisation de modèles logiques de documents multimédias (structuration du document indépendamment de sa présentation), de par l'expérience acquise dans l'édition de documents textuels semble une voie à explorer, voie qui à ce jour est complètement inexploitée par les outils actuels qui se contentent d'offrir des modèles de présentation.

4.2 Les environnements de programmation

Nous avons vu dans la section précédente que les langages de programmation permettaient à l'auteur un grand pouvoir d'expression ; la contrepartie est liée aux compétences en informatique nécessaires :

- Problèmes d'inaccessibilité aux non-informaticiens.
- Problèmes de mise au point, l'auteur doit utiliser des débogueurs classiques, le contexte d'utilisation (c'est-à-dire l'écriture de documents multimédias) n'est pas exploité. Par exemple, on pourrait imaginer des modules spécialisés pour aider l'auteur dans ses spécifications de synchronisations spatiales et temporelles.
- Absence de WYSIWYG, c'est-à-dire qu'avant de visualiser son document, l'auteur doit le compiler.

Cependant des techniques existent pour réduire ces difficultés, on peut citer, par exemple, la programmation

visuelle (section 4.2.1) et la programmation par démonstration (section 4.2.2).

4.2.1 Les langages de programmation visuelle

Des travaux ont été réalisés pour faciliter l'accès à la programmation à des non-informaticiens. L'objectif de la programmation visuelle est de fournir une interface pour décrire de façon simple et plus conviviale des programmes. On peut citer comme exemples d'environnements Venn [DelBimbo96] et dans le cadre des documents multimédias Icon Author [AimTech96]. Dans l'exemple de la Figure II-15, on peut voir un exemple de spécification de document. L'auteur a défini une boucle avec une succession de calculs puis un affichage d'objets qui sera interrompue par un clic de l'utilisateur. L'auteur édite son document au travers de palettes d'opérateurs, et pour chaque opérateur, il a la possibilité de définir un certain nombre de paramètres.

Cependant on peut noter qu'en facilitant la tâche de l'auteur, ces méthodes limitent l'expressivité du langage et le pouvoir de l'auteur. En effet, on peut difficilement spécifier de gros programmes et/ou des programmes complexes par ce biais du fait des difficultés de perception liées à l'affichage et au volume d'information que l'on peut visualiser.

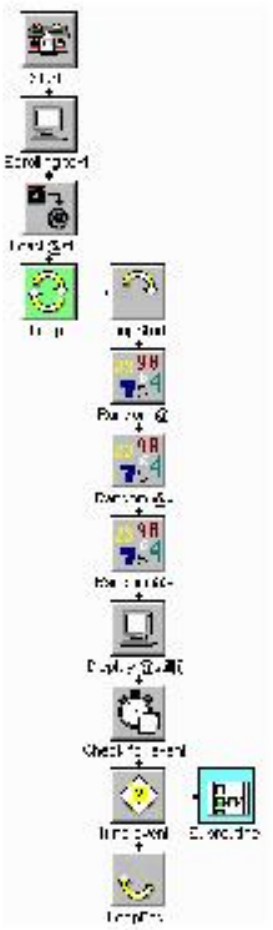


Figure II-15 : Logigramme d'un programme réalisé avec IconAuthor

Les limitations des environnements de programmation visuelle sont du même ordre que pour les programmes classiques :

- Difficulté d'écrire de gros programmes/documents. L'auteur a du mal à avoir une perception globale du comportement de son document. De plus, peu de techniques ont été développées pour améliorer l'affichage et permettre une meilleure visualisation des informations.
- Difficulté de description de comportements complexes. La description d'un gros document entraîne une complexité visuelle difficilement manipulable pour un auteur.
- Difficulté de spécification de document réactif. Il est difficile pour un auteur de manipuler un

document contenant de nombreuses branches, de par la complexité visuelle de la représentation.

4.2.2 La programmation par démonstration

L'objectif de la programmation par démonstration est de faciliter la spécification de comportements complexes ou répétitifs. Par exemple, l'auteur exécute une fois la tâche, et le système automatise la tâche par le biais de macros. L'utilisateur, quand il aura besoin d'exécuter une autre fois la tâche, fera appel à la macro. Dans le cadre de Toolbook [Asymetrix99] l'utilisateur peut montrer un déplacement d'objet à l'écran à l'aide de la souris, et le système écrit pour l'auteur le code correspondant dans les langages de scripts de Toolbook. Actuellement, la programmation par démonstration vise donc à aider l'auteur dans l'écriture de scripts. Elle pourrait être, par exemple, utilisée pour l'aider à spécifier le comportement temporel de son document. Par exemple, dans Director, pour spécifier un déplacement complexe, l'auteur peut dessiner à l'aide de fonctions prédéfinies le déplacement de son objet, le système en déduira le placement de l'objet réel à chaque instant. Ces techniques fournissent des idées pour la phase de visualisation ainsi que sur les modes d'action pour l'édition. Ces méthodes sont souvent utilisées en complément dans des outils plus complets (Toolbook).

4.2.3. Synthèse sur les environnements de programmation

Les environnements de programmation sont encore réservés aux informaticiens. L'utilisation d'outils utilisant des métaphores graphiques ou des méthodes de programmation par démonstration, facilite l'accès des non-informaticiens à de tels outils. Cependant, pour atteindre cet objectif, ces outils réduisent le pouvoir d'expression de l'auteur. De plus, un autre inconvénient de ces approches est qu'il n'existe pas d'aide pour l'auteur de documents multimédias (librairies, modèles, classes,).

Cependant, on peut imaginer des améliorations possibles de ces techniques en fournissant à l'auteur un ensemble de classes permettant de définir des fragments de documents. De plus, on pourrait imaginer une extension des techniques de programmation par démonstration, de manière à permettre au système de déduire des relations à la place de simplement noter des propriétés absolues sur les médias. Ces extensions peuvent s'inspirer des techniques développées dans Garnet [Sannella92] qui reposent sur des mécanismes à base de contraintes pour reconnaître des propriétés. Cela faciliterait la relecture des programmes produits et rendrait le document généré plus flexible.

4.3 Les environnements auteur complets

La troisième catégorie de systèmes auteur laisse plus de latitude à l'auteur, tout en offrant des moyens de l'aider dans la tâche complexe qu'est la création de documents. Ces outils sont dits complets, car ils permettent à l'auteur de spécifier toutes les caractéristiques du document à toutes les étapes du processus d'édition. Ces outils mélangent le plus souvent les formalismes d'édition, pour permettre différents moyens de spécification.

4.3.1 Director

Director est un des premiers outils dédié à l'édition de documents multimédias. Il est apparu à la fin des années 80 et ne cesse de changer pour intégrer les évolutions des documents et améliorer la phase d'édition.

La phase d'édition dans Director est basée sur quatre étapes :

- *Création d'un objet de base* : cet objet peut être une image, un texte, un son.
- *Placement temporel d'un objet* : il s'effectue sur une vue temporelle (Figure II-16) où l'auteur place de façon absolue l'objet. C'est-à-dire qu'il spécifie explicitement l'instant de début de son objet et la durée de celui-ci. Ce type de placement ne permet pas de définir des synchronisations évoluées entre les objets ni de faire des placements relatifs d'un objet par rapport à un autre.

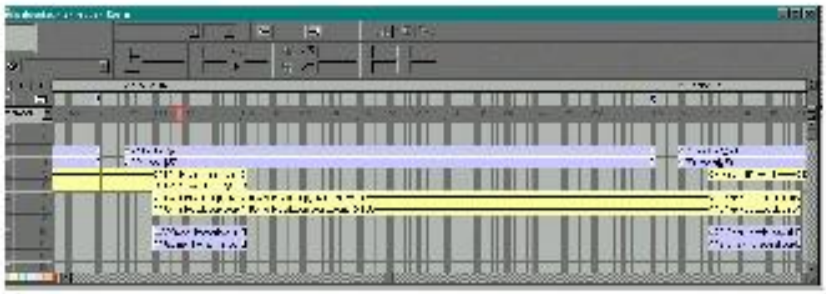


Figure II-16 : Vue temporelle de Director

- *Placement spatial d'un objet* : le placement spatial s'effectue de façon absolue dans la fenêtre de présentation (Figure II-17).

Figure II-17 : : Vue de présentation de Director

- *Description fine de la synchronisation temporelle* et des événements liés aux interactions des utilisateurs : cette description se fait au moyen d'un langage de programmation (Lingo [Macromedia-Lingo00]) dont on peut voir un exemple dans la Figure II-18. Dans cet exemple, on peut voir un comportement défini pour garder une scène à l'écran. Les scripts sont par exemple utilisés pour définir un bouton qui permettra de déclencher le début d'une vidéo. L'utilisateur peut associer un comportement à plusieurs types d'événements : clic utilisateur, changement de comportement d'un objet. L'utilisation de scripts augmente la complexité de la spécification dans Director. En effet, même si le langage de scripts est relativement simple, une utilisation évoluée nécessite des compétences informatiques.

```
--------------------------------------------------------------------------------
-- Attente
-- Ce script permet de garder affiché la scène courante, pendant Howlong secondes.
--------------------------------------------------------------------------------

property howLong, currentTime --déclaration des variables
  on prepareFrame -- événement associé à la création de la scène
  currentTime = the timer
end

on exitFrame --événement associé à la fin de la scène
  repeat while the timer < currentTime + howlong
    nothing
  end repeat
end

-- mise à jour des propriétés et initialisation du vérificateur de propriétés.
on getPropertyDescriptionList
  propertyDescriptionList = [ ¬
  #howLong: [ #comment: "durée d'attente (1/60e seconde):"
  #format: #integer,
  #default: 10
  return propertyDescriptionList
end

on getBehaviorDescription
  return "garde la scène présente à l'écran en fonction de la durée spécifiée."
End
```

Figure II-18 : Exemple de script dans Director

À tout moment de l'édition, l'auteur peut visualiser les objets présents dans son document (Figure II-19). De plus lors des opérations d'édition, le système donne des informations complémentaires à l'auteur : par exemple dans la Figure II-17, lorsque l'auteur manipule un objet dans la vue de présentation, le système affiche des informations : coordonnées absolues de l'objet ainsi qu'un conseil pour réaliser une opération.

Figure II-19 : Vue objet de Director

Director est un environnement d'édition complet, qui, à partir des informations auteur, génère un fichier compilé, qui servira de support à l'exécution du document. Ce fichier contient des informations spécifiques au système d'exploitation pour lequel il est destiné. Cette approche limite donc la portabilité du document produit. Ce qui dans le cas de Director limite la diffusion du document, étant donné qu'il n'existe pas de machine de présentation sur tous les systèmes d'exploitation. On peut noter que l'environnement ne fournit aucun moyen d'accéder aux informations liées aux périphériques de présentation. Il est donc impossible à l'auteur de faire des documents adaptables sans intervention du lecteur.

Par rapport aux différents formalismes présentés précédemment, Director offre donc à la fois une édition via un formalisme absolu et via un formalisme à base d'un langage de programmation. Cette double combinaison est intéressante du point de vue de l'expressivité, néanmoins dans les deux cas, elle rend difficile la maintenance du document au cours du cycle d'édition.

Du point de vue du cycle d'édition, l'aspect important dans Director est la présence d'une vue temporelle qui permet à l'auteur d'avoir une idée des enchaînements de son document.

4.3.2 Grins

Grins est un éditeur de documents SMIL qui est issu des travaux menés par l'équipe du CWI d'Amsterdam autour de CMIFed [Hardman93]. Il est en cours de commercialisation par la société Oratrix. Grins permet à l'auteur de spécifier de manière indépendante les informations spatiales et temporelles.

Cet éditeur est composé de quatre vues :

- Une vue temporelle hiérarchique (Figure II-20a) qui permet de spécifier la structure temporelle du document : cette vue permet de représenter la hiérarchie temporelle sous forme de boîtes englobantes, l'axe vertical servant à représenter le placement temporel des objets à l'intérieur d'un noeud : en séquence (les objets sont affichés les uns en dessous des autres) ou en parallèle (les objets sont affichés les uns à côté des autres).
- Une vue des régions (Figure II-20b) pour définir le placement spatial des objets.
- Une vue temporelle (Figure II-21) qui est générée à partir des informations de la vue temporelle hiérarchique et des attributs que l'auteur a pu placer sur les différents médias.

- Une vue de présentation pour exécuter le document.

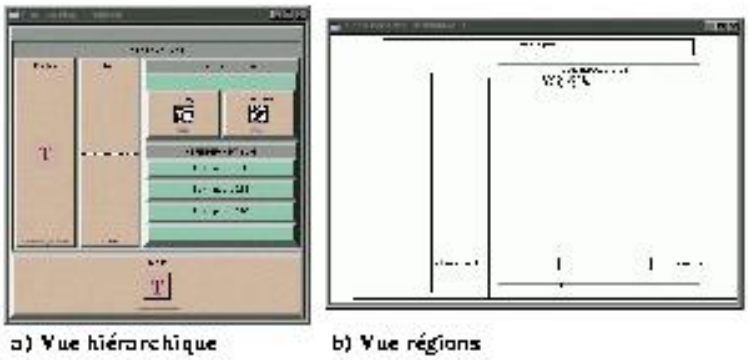


Figure II-20 : Grins : vue hiérarchique et vue des régions

L'édition au sein de Grins se fait essentiellement en 3 étapes :

- Spécification du placement temporel et spatial dans les vues adaptées.
- Spécification d'attributs à l'aide de formulaires.
- Visualisation de la dimension temporelle dans la vue temporelle et exécution du document. L'auteur peut de plus, positionner des arcs de synchronisation entre les objets.

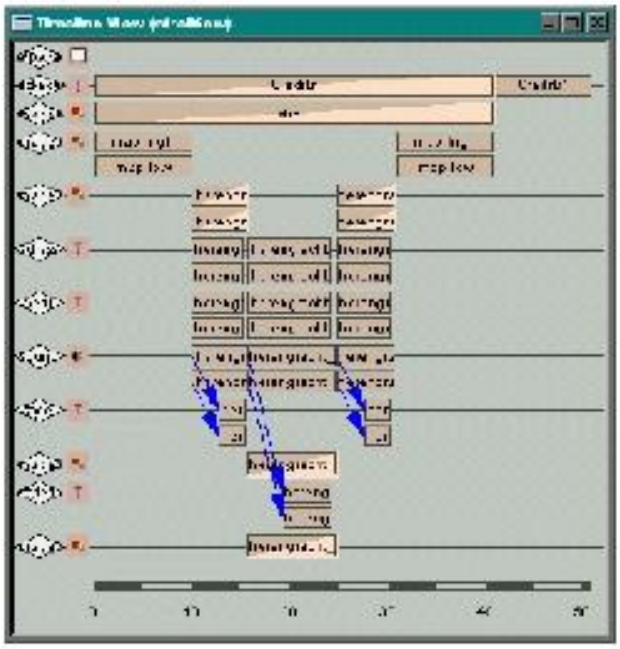


Figure II-21 : Grins, vue temporelle

Grins permet de spécifier un document de façon relative, avec la possibilité de définir des événements entre les objets (dans la limite de ce que permet SMIL 1.0).

Par exemple, l'édition temporelle qui est basée sur une structure d'arbre d'opérateurs se fait dans une vue hiérarchique, le placement temporel résultant sera ensuite visualisé dans une vue temporelle qui représente sur des axes horizontaux le placement des objets. L'auteur peut à ce moment raffiner sa spécification à l'aide d'arcs de synchronisation. Ils permettent de lier le début ou la fin d'un objet au début ou à la fin d'un autre objet (c'est équivalent au placement d'attribut *begin* et *end* relatif).

Du point de vue de l'auteur, on ne peut pas manipuler la vue temporelle pour ajuster directement le placement des objets en modifiant ainsi interactivement les arcs de synchronisation et la valeur des attributs *begin* et *end*. Notons que Grins n'offre pas de service d'aide au formatage, l'auteur doit donc spécifier explicitement les durées sur les objets.

On peut aussi remarquer que le formalisme d'édition de Grins est complètement lié à celui du langage SMIL, ce qui par exemple, pour la dimension spatiale, limite beaucoup les facilités d'édition de l'auteur qui est, dans

ce cas, obligé de placer les objets de manière absolue.

Les deux aspects importants de l'édition offert par Grins sont la visualisation de la structure d'opérateurs combinée avec une visualisation des informations temporelles dans la vue hiérarchique, ainsi que l'édition des attributs spatiaux par manipulation directe.

4.3.3 Madeus-97

Madeus, éditeur développé au sein du projet Opéra, permet d'éditer des documents multimédias spécifiés à l'aide de relations spatiales et temporelles entre les objets. La première version de Madeus (en 1997) a été réalisée par Nabil Layaïda et Loay Sabry au cours de leurs thèses [Layaïda97, Sabry99]. Depuis, ce prototype ne cesse d'évoluer et nous en présenterons une version récente qui inclue les travaux de cette thèse dans le chapitre VI.

Nous allons présenter dans cette section la version de Madeus au début de cette thèse. L'environnement auteur se compose essentiellement d'une vue présentation (Figure II-22 a) et de palettes (Figure II-22 b et Figure II-22 c), qui permettent à l'auteur de spécifier des relations temporelles ou spatiales entre les objets du document, après les avoir sélectionnés dans la vue de présentation. La vue de présentation est à la fois une vue dans laquelle l'auteur peut jouer son document et aussi une vue d'édition puisque l'auteur peut définir le placement des objets en déplaçant ceux-ci directement à l'écran.

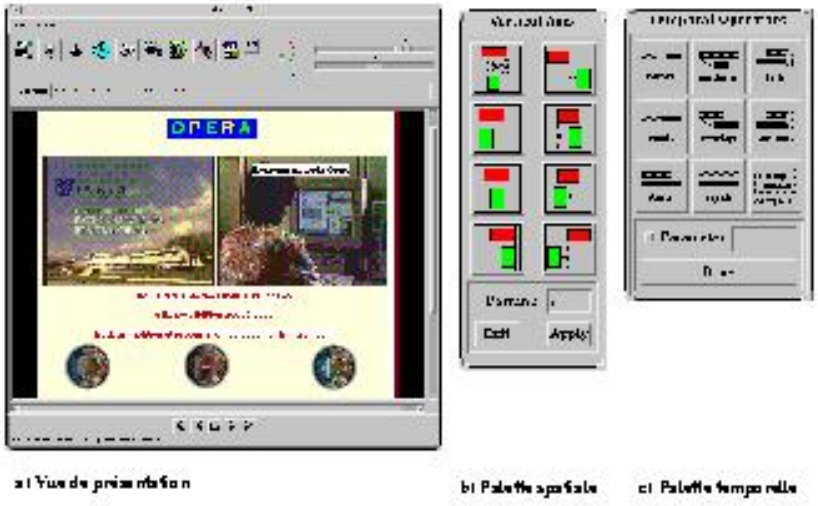


Figure II-22 : Les différentes vues de l'environnement Madeus-97

L'objectif était d'intégrer le WYSIWYG le plus possible tout au long du processus d'édition. L'environnement auteur de Madeus est une des premières réalisations d'un environnement d'édition /présentation intégré.

Dans la version Madeus-97, les principales lacunes du point de vue de l'édition sont :

- Absence de vue temporelle. La perception de l'enchaînement temporel se fait de manière locale par la vue de présentation.
- Absence d'un module de formatage temporel complet qui permettrait de calculer automatiquement le placement et la durée des objets temporels.
- Impossibilité d'exprimer des structures de contrôle avec de l'inderminisme.

Nous verrons que nous apporterons par la suite une réponse aux deux premiers points.

4.4 Synthèse sur les environnements d'édition

Dans cette section, nous avons pu voir qu'il existait aujourd'hui de nombreux outils auteur de documents multimédias. Cependant ces outils ne permettent pas de fournir un bon compromis entre : pouvoir d'expression et simplicité d'édition, même si un outil comme Grins offre des services d'édition avec des services de visualisation d'informations temporelles qui semblent intéressants et prometteurs. Parmi les

systèmes d'édition, l'auteur a aujourd'hui le choix entre :

- Des outils à base de modèles prédéfinis qui sont relativement simples d'utilisation et qui permettent à l'auteur d'écrire des documents proches de modèles, ou d'exemples prédéfinis. Cependant, ces outils autorisent difficilement un auteur à faire ce qu'il désire indépendamment des exemples proposés.
- Des outils de plus haut niveau, basés sur un formalisme proche du langage de sauvegarde, comme Grins et Madeus. Ces outils doivent cependant améliorer les aides qu'ils fournissent à l'auteur pour l'édition.
- Des outils mélangeant plusieurs formalismes de spécification en fonction des besoins de l'auteur. C'est le cas par exemple, de Director qui offre un formalisme absolu et un formalisme de programmation pour répondre aux besoins de l'auteur. Cette approche semble la plus prometteuse, mais un choix judicieux sur les formalismes proposés doit être fait.

5 Conclusion

5.1 Bilan des approches de spécification et d'édition

On s'aperçoit aujourd'hui qu'il existe un lien très fort entre le formalisme d'édition et les facilités offertes à l'auteur. L'auteur a le choix entre :

- Spécifier son document de manière absolue à l'aide d'une interface de type vue temporelle. L'expressivité de ce genre d'approche est relativement faible. L'auteur ne peut que placer dans le temps les objets et ne peut pas définir de relations entre eux. Par contre, les interfaces fournies sont relativement simples d'utilisation (SmilWizard, RealSlideShow, Powerpoint).
- Spécifier son document à l'aide d'un langage de programmation. Ce choix offre une très grande expressivité à l'auteur, par contre l'interface revient à des environnements de programmation classique, qui sont difficilement exploitables pour des non-informaticiens (C++/ Java).
- Spécifier son document à base d'approches relationnelles. Dans ces approches, l'objectif est d'offrir le meilleur compromis entre l'expressivité et la simplicité d'édition puisque l'auteur définit des relations entre les objets. Le système d'édition doit en contrepartie fournir des mécanismes de visualisation plus complexes pour aider l'auteur à manipuler les relations et à comprendre et maîtriser l'espace de solutions.

De l'étude des sections précédentes, on peut en conclure le récapitulatif de la Figure II-23. La partie édition des formalismes a été obtenue en se basant sur un environnement (fictif ou réel), manipulant directement les concepts de l'approche. Par exemple une édition basée sur une vue hiérarchique, dans le cas d'arbre d'opérateurs, ou une édition via une vue temporelle pour les approches absolues,

De cette figure, on peut noter les éléments suivants :

- *Formalismes* :
  - absolu : ce formalisme offre une édition pauvre mais élémentaire.
  - programmation : ce formalisme très riche n'offre que peu de possibilités d'aide dans le cadre de l'édition. La vie du document est liée au coût d'écriture.
  - événementiel : il offre une forte expressivité mais la spécification rend difficile la maintenance du document.
  - relationnel : il offre une forte expressivité mais nécessite une aide à la visualisation et au formatage de la part du système auteur.
- *Outils*:
  - Director : il utilise les formalismes absolu et de programmation. De ce fait il cumule les avantages et les inconvénients de ces deux approches.
  - SmilWizard : simple d'utilisation, cependant l'auteur n'a aucun contrôle sur ce qu'il produit.

- Grins : basé sur un arbre d'opérateurs temporels (celui de SMIL), cet outil est très dépendant du modèle sous-jacent. L'auteur devra, lors de chaque modification de la présentation de son document, changer sa structure de document du fait que celle-ci est liée fortement à la présentation.
- Madeus-97 : basé sur une approche relationnelle, cet outil était une première étape dans la réalisation d'un environnement auteur complet. Il a de ce fait des lacunes dans l'aide à la visualisation de l'auteur et l'automatisation de certaines tâches.

Les parties grisées sont les parties qui nous semblent les plus significatives.

Besoins	**Approches**							
	Formalismes				**Outils**			
	Absolu	Progr.	Evén.	Relation	Director	Smil Wizard	Grins	Madeus-97
Expressivité								
Médias	+	++	++	++	++	+	+	+
Interactions	-	++	++	+	++	-	+	+
Simplicité								
Instructions	-	++	++	+	++	-	+	-
Structure	-	+	+	++	-	-	+	++
Adaptation	-	++	++	++	-	-	+	++
Modification	-	-	-	++	-	-	-	+
Edition								
Visualisation	++	-	-	+	+	+	+	+
Fonction d'édition	-	-	+	++	-	+	+	+
Présentation	+	+	+	++	+	+	+	+
Cohérence	++	-	-	++	++	++	+	++
Cycle d'édition	-	-	-	++	-	-	+	++
Vie du document	-	-	+	++	-	-	-	-

Figure II-23 : Récapitulatif de la satisfaction des différents besoins

5.2 Choix d'un formalisme d'édition

Aujourd'hui, tous les formalismes de spécification de documents multimédias permettent de définir des langages avec à peu près le même pouvoir d'expression si l'on considère que tous les objets sont déterministes.

On peut en effet, spécifier tous les documents sous une forme absolue avec des liens. Pour cela, on explicite toutes les exécutions possibles, exécutions qu'on relie par des liens. On peut noter que de tels documents contiennent plusieurs présentations possibles suivant les liens parcourus. De ce fait le critère de pouvoir d'expression est peu discriminant dans la réalisation d'un environnement auteur si l'on considère des

documents indéterministes avec navigation. L'aspect important est donc la facilité de description du comportement temporel et spatial.

Au cours de cette thèse, nous nous intéresserons essentiellement aux documents déterministes avec de la navigation (section 2.1.1). Au cours des chapitres suivants nous étudierons l'édition et les services que peut fournir un système d'édition pour aider l'auteur dans ce contexte. Nous ferons à la fin de la thèse des propositions pour généraliser les travaux réalisés aux documents contenant de l'indéterminisme.

5.3 Bilan des besoins non satisfaits

Parmi les besoins peu satisfaits à l'heure actuelle, ceux qui nous semblent les plus importants dans un processus d'édition complet, sont la facilité de modification du document et la visualisation des informations contenues dans le document. Les approches relationnelles semblent offrir aujourd'hui les meilleures capacités pour répondre à ces besoins, de par la flexibilité intrinsèque de ces approches. Cependant les outils actuels n'offrent pas de facilité d'édition pour exploiter cette souplesse. Par exemple, aucun système auteur n'offre une édition par manipulation directe avec maintien des relations.

De plus, les outils actuels sont souvent des éditeurs de langage sans être réellement des outils auteur, car ils n'offrent que peu d'aide à l'auteur et qu'ils n'automatisent que peu de tâches (formatage, détection des incohérences).

Chapitre III : Spécification d'un environnement auteur idéal

1. Introduction

Au cours du chapitre précédent, nous avons pu voir les différents formalismes utilisés pour définir des documents multimédias, les types d'environnement d'édition qui existent, ainsi que les différents besoins de l'auteur au cours du processus d'édition. Nous avons vu comment ces besoins étaient plus ou moins couverts par les environnements existants. À partir de ces informations, nous allons à présent définir ce que nous considérons être l'environnement d'édition de documents multimédias idéal, c'est-à-dire celui satisfaisant au mieux les besoins des auteurs identifiés auparavant.
Pour cela nous nous inspirerons des différents systèmes auteur de documents multimédias présentés, mais ces derniers n'apportant pas une réponse satisfaisante à tous les besoins, nous élargirons notre domaine de référence pour chercher des idées de solutions dans des domaines voisins comme l'édition de texte, d'hypertexte ou de vidéo. Dans ce chapitre, nous allons essentiellement nous intéresser aux fonctionnalités que doit fournir un tel environnement. Pour le moment, nous ne nous intéresserons pas ou peu aux aspects graphiques de l'interface. Ces aspects ne sont pas indépendants, mais nous nous focaliserons sur les fonctions intrinsèques de base.
Dans un premier temps, nous présenterons les principes généraux de cet environnement (section 2).
Nous validerons ensuite cette proposition d'environnement idéal en indiquant comment il permet de répondre aux différents besoins de visualisation (section 3), ainsi qu'aux différents besoins de navigation de l'auteur (section 4). Et nous terminerons ce chapitre par l'illustration de l'utilisation de cet environnement auteur au travers d'une session d'édition (section 5).

2. Spécification de l'architecture de l'environnement idéal

Nous allons présenter dans un premier temps l'architecture générale de cet environnement d'édition idéal (section 2.1). Dans un deuxième temps, nous spécifierons le formalisme d'édition que nous utiliserons dans cet environnement d'édition (section 2.2). Dans un troisième temps nous nous intéresserons aux modes de visualisation que cet environnement doit fournir (section 2.3) et enfin dans une dernière partie nous présenterons les différents services d'aide que devra fournir cet environnement pour faciliter la tâche d'édition de l'auteur (section 2.4).

2.1 Rappel du cycle d'édition et architecture générale

Avant d'aller plus loin, nous rappelons le cycle d'édition défini dans le chapitre précédent (voir chapitre II section 2.2.1), ce processus se décompose suivant cinq étapes :

1. Ouverture du document ;
2. Visualisation du document et navigation à l'intérieur des différentes informations ;
3. Modification du document ;
4. Vérification de la cohérence et formatage ;
5. Sauvegarde du document.

De ce processus de base, on a proposé une architecture générale de l'environnement auteur qui se découpe en quatre grandes fonctions (voir chapitre II section 2.2.1 et Figure III-1) :

- Gestion de fichier ;
- Visualisation d'information ;
- Opération d'édition ;
- Aide à l'auteur.

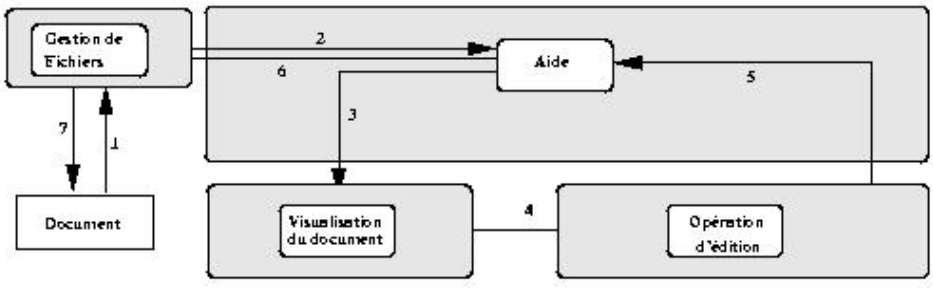


Figure III-1 : Cycle d'édition simple

Dans le chapitre précédent nous avons proposé cette architecture générale pour un environnement d'édition de documents multimédias. L'environnement auteur idéal que nous allons proposer est basé sur cette architecture. Au cours de ce chapitre, nous allons nous intéresser essentiellement à la définition du formalisme d'édition de cet environnement. Ce formalisme est le coeur de l'environnement d'édition. Nous nous intéresserons dans un deuxième temps aux services d'aide à l'édition que doit fournir un environnement auteur.

2.2 Spécification d'un formalisme d'édition

Nous allons dans un premier temps préciser quel type de formalisme d'édition nous allons choisir et justifier notre choix (section 2.2.1). Dans un deuxième temps, nous présenterons ce formalisme d'édition (section 2.2.2), et enfin, dans une troisième partie, nous identifierons les services que devra fournir l'environnement auteur idéal pour éditer ce formalisme (section 2.2.3).

2.2.1 Motivations

Comme nous avons pu le voir dans le chapitre précédent (section 3), il existe de nombreux langages pour décrire des documents multimédias.

La première solution pour réaliser un environnement auteur consiste à utiliser le formalisme du langage de sauvegarde/présentation comme formalisme d'édition. Néanmoins ces langages sont souvent adaptés à la présentation de documents multimédias, mais ils sont en général peu adaptés à l'édition.

Une deuxième approche consiste à réaliser un environnement avec son propre formalisme d'édition et de réaliser des fonctions d'import / export vers d'autres langages. Cependant, l'inconvénient de telles fonctions, est que l'on perd souvent les informations d'édition. En effet, le fichier de restitution essaie de conserver au maximum les informations de présentation, en oubliant généralement les informations qui ont permis d'arriver à cette présentation. Cela pose principalement deux problèmes : l'édition par un tiers du fichier de restitution, et le problème de stockage et de gestion de version des deux fichiers qu'il faut garder (sauvegarde et restitution). Cette situation arrive par exemple, quand une personne écrit ses pages Web avec Word pour les exporter finalement en HTML.

Cette deuxième solution, serait celle qui offrirait la plus grande satisfaction pour le concepteur de l'environnement auteur. Cependant, la diversité des langages actuels est tel que cela est impossible. En effet, le formalisme commun nécessaire serait une combinaison des différents formalismes, et ne permettrait pas de répondre aux besoins des auteurs vis-à-vis de l'édition. En effet, on ne peut pas éditer de la même façon tous les langages, cela ne serait pas pertinent vis-à-vis des formats existants aujourd'hui.

Une solution autre consiste à faire un compromis entre ces deux approches.

Cette solution est illustrée dans la Figure III-2. Elle consiste à faire cohabiter les deux solutions précédentes, c'est-à-dire faire cohabiter à la fois le formalisme de haut niveau proposé par l'environnement, tout en laissant la possibilité à l'auteur d'accéder au langage de présentation.

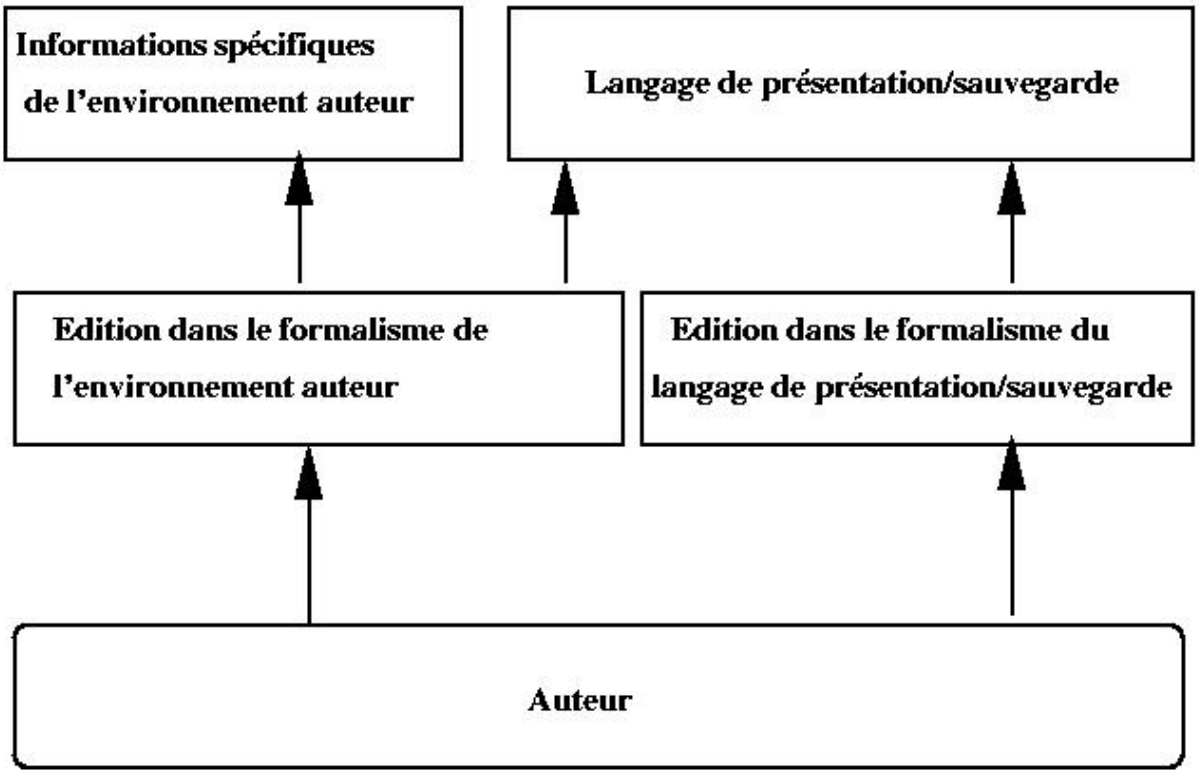


Figure III-2 : Un environnement auteur utilisant plusieurs formalismes

Dans ce cas-là, c'est le système qui s'occupe de conserver la relation entre le formalisme de l'environnement et celui du langage de présentation. Il est nécessaire pour cela de stocker certaines informations complémentaires du fait qu'il n'y ait pas de bijection entre les possibilités du langage et celui de l'environnement. Ces informations complémentaires pourront être stockées à l'intérieur même du fichier par exemple par l'intermédiaire des espaces de noms proposés dans XML. Ainsi, dans l'exemple de la Figure III-3 on peut voir un exemple de fichier SMIL où l'environnement auteur a sauvegardé des informations spécifiques (des relations spatiales).

La manipulation de ces deux formalismes peut rendre plus complexe la tâche de l'auteur. L'environnement doit aider l'auteur dans cette compréhension en offrant par exemple des services de visualisation adaptés.

Nous verrons en conclusion de ce chapitre en quoi ce double formalisme permet de répondre aux différents besoins auteur.

```
<?xml version="1.0"?>

<smil>
<head>
<layout xmlns:EnvAuteur="http//www.inrialpes.fr/opera/EnvA.dtd">
      <region id="région1" top="180" left="80" width="100" height="40" />
      <region id="région2" top="240" left="100" width="100" height="40" />
      <region id="région3" top="300" left="120" width="100" height="40" />
      <region id="région4" top="360" left="140" width="100" height="40" />
      <region id="région5" top="420" left="160" width="100" height="40" />
      <EnvAuteur:Relation>
      <EnvAuteur:LeftAlign EnvAuteur:el1="région1" EnvAuteur:el2="région2">
      </EnvAuteur:Relation>
</layout>
</head>
<body>

<par dur="50" id="Document" >
      <text src="/home/Texte1.html" id="T1" region="region1"/>
      <text src="/home/Texte2.html" id="T2" begin="10.0" region="region2"/>
      <text src="/home/Texte3.html" id="T3" begin="20.0" region="region3"/>
      <text src="/home/Texte4.html" id="T4" begin="30.0" region="region4"/>
      <text src="/home/Texte5.html" id="T5" begin="40.0" region="region5"/>
</par>
</body>
</smil>
```

Figure III-3 : Exemple de fichier utilisant un espace de noms

De plus, l'importance réciproque de ces besoins dépend de l'objectif de l'auteur. On peut définir trois classes d'objectifs lors de la conception d'un document :

- *simplicité d'édition et diffusion*: tout auteur souhaite pouvoir éditer simplement ses documents, pour répondre à cette attente l'environnement auteur idéal fournit un formalisme de haut niveau qui permet de simplifier la phase d'édition. De plus, la sauvegarde des informations d'édition dans le fichier de présentation permet à l'auteur de ne pas avoir le problème de gestion de version entre les différentes formes de son document, problème rencontré dans les approches utilisant des fonctions d'export pour diffuser le document.
- *partage entre plusieurs auteurs du document* : l'environnement auteur basé au dessus d'un standard de spécification garantit à l'auteur une portabilité de son document tout en permettant à d'autres personnes d'éditer son document. Ce besoin arrive essentiellement dans un contexte d'édition du document à plusieurs où l'utilisation d'un format commun impose des contraintes entre les différents utilisateurs.
- *simplicité d'édition, diffusion et partage du document produit*: ce besoin qui est la combinaison des deux premiers est satisfait par le double formalisme offert. En effet, celui-ci permet à l'auteur d'utiliser un standard de spécification, tout en utilisant partiellement les facilités de spécification de l'environnement pour simplifier sa tâche.

En conclusion, le choix que nous allons faire est de proposer un formalisme d'édition propre à l'environnement tout en laissant la possibilité à l'auteur d'accéder au formalisme du langage.

2.2.2 Le formalisme d'édition de l'environnement auteur idéal

Le choix du formalisme que nous allons employer est motivé par l'analyse du chapitre précédent. En effet, des formalismes comme le formalisme absolu ou événementiel sont trop complexes et rendent le document difficilement gérable de manière globale. De plus, les travaux menés avec l'équipe Airelle [Bétrancourt99] ont permis de montrer que l'approche basée sur les groupes et les relations était plus facilement compréhensible et offrait une édition plus souple.

De ces faits nous allons proposer un formalisme d'édition relationnel caractérisé par :

- La spécification des informations temporelles et/ou spatiales par relations, même pour des langages non relationnels.
- La spécification des médias sous forme d'intervalle de valeurs comme dans les langages (ISIS, Madeus).

Cependant, comme nous venons de le voir dans la section précédente le formalisme d'édition offert par l'environnement d'édition est là pour faciliter l'édition du langage de présentation. Pour chaque environnement d'édition réalisé, le formalisme d'édition est donc dépendant du langage de présentation sur lequel sera basé l'édition. Néanmoins, quel que soit le langage de présentation et de sauvegarde utilisé les environnements d'édition auront un certain nombre de caractéristiques communes. Nous allons donc présenter ses caractéristiques communes en trois étapes : définition des médias, définition des groupes et définition des relations, tout en spécialisant nos propos dans le cadre de langages de présentation précis lorsque cela sera nécessaire. Les langages qui nous servirons à illustrer ce chapitre seront SMIL, MHML et Madeus. Ces langages sont aujourd'hui représentatifs des approches événementielles et relatives de spécifications.

Le formalisme fourni par l'environnement auteur est le suivant :

Définition de médias :

Les médias sont caractérisés par un ensemble d'attributs :

- Le type du média.
- Les attributs spatiaux du média: X, Y, Hauteur, Largeur.
- Les attributs de présentation du média : couleur, fonte, police.
- Les attributs temporels : début, fin, durée.
- Le lien de navigation associé au média, ce lien permet au lecteur de naviguer de document en document.
- Le lien temporel associé au média, un lien temporel permet d'aller à un instant donné de la présentation : dans l'exemple de la Figure III-4, on visualise une exécution d'un document multimédia dans lequel on a spécifié le lien temporel : "Si l'auteur clique sur l'objet A, alors on saute à la 40ème seconde du document". Le comportement de ce lien est le suivant : si le lecteur clique sur l'objet A à la 20ème seconde de la présentation, cela aura pour effet d'interrompre tous les objets en cours (A, D et E) et de déclencher la présentation de tous les objets

présents à la 40ème seconde (B et C). Dans le cas de l'objet B, celui-ci sera déclenché à sa 5ème seconde du fait qu'il aurait normalement dû commencer à la 35ème seconde du document.

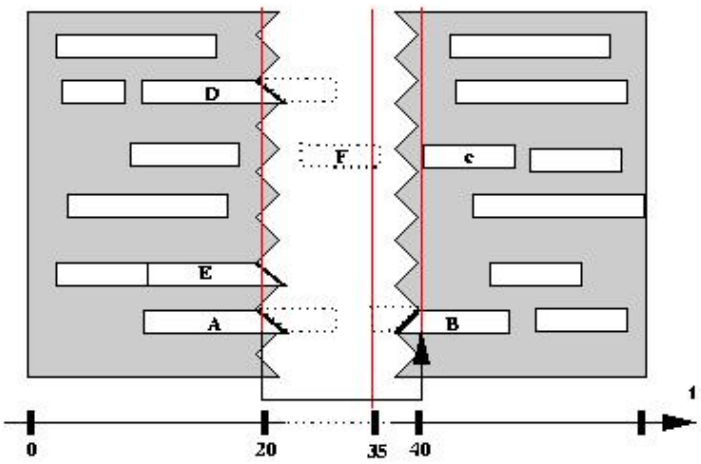


Figure III-4 : Lien temporel

On peut noter que la spécification de la valeur d'un attribut n'est pas obligatoire. De plus, chaque attribut, dont l'ensemble de valeurs est énumérable est défini par intervalle de valeurs possibles, par exemple :

- Début = [13, 15, 18], qui signifie que le début de l'objet est compris entre 13 et 18 secondes et l'auteur indique qu'il aimerait qu'il commence à la 15e seconde si cela est possible ;
- X = [100, ? ?], qui signifie que l'auteur indique que l'objet doit au moins être à 100 unités du bord.

Dans l'exemple de la Figure III-5 nous donnons un exemple de syntaxe (décrite en XML) pouvant servir pour décrire un média.

```
<Média Type="Image" >

<AttributSpatiaux
       X="8 10 12"
       Y="10 10 14"
       Hauteur="12"
       Largeur="10">
<AttributTemporels
       Begin="12 14 15"
       End="16 16 16"
       Durée="1 2 4">
<LienNavigation =http://www.inrialpes.fr/opera/>
<LienTemporel ="#14s">
</Media>
```

Figure III-5 : Exemple de syntaxe décrivant un media

Définition de la présentation :

Nous venons de voir comment définir un élément média, cet élément est la base de tout document multimédia. Nous allons maintenant nous intéresser à la définition de la présentation du document. C'est-à-dire à l'agencement des différents médias temporellement et spatialement. Cet agencement se fera dans le cadre d'une structure à base de groupes. Nous allons dans un premier temps écrire les attributs associés à un groupe avant de décrire plus précisément les groupes temporels et spatiaux.

Un groupe est caractérisé par :

- Un ensemble d'éléments qui composent ce groupe.
- Un opérateur qui permet de définir un comportement pour tous les éléments.
- Liste de relations entre les objets. La liste des relations permises est dépendante du langage de présentation/sauvegarde.
- Définition d'attributs de présentation :
  - attributs : X, Y, Hauteur, Largeur pour les groupes spatiaux.
  - attributs : début, fin, durée pour les groupes temporels.

Nous allons maintenant donner un exemple de syntaxe pouvant permettre de décrire cette spécification de la présentation (Figure III-6).

```
<GroupeTemporel
      ListeElement="A B C D E"
      Opérateur ="Inside" >
      <AttributsTemporel
            Début = "14 18 19"
            Durée = " 5 6 8"
            fin = "19 24 27" />
      <ListeRelations >
            <Relation Nom="Before" Objet1="A" Objet2="B" Param="4s">
      </ListeRelations>
</GroupeTemporel>
```

Figure III-6 : Exemple de syntaxe permettant de décrire la présentation

Le formalisme que nous venons de définir permet d'associer à chaque groupe un opérateur, un ensemble de relations et des attributs. Cet ensemble d'informations permet de définir le placement relatif de chaque objet du groupe par rapport au début du groupe. Le placement relatif d'un objet est donc le résultat d'un calcul (le formatage) entre ces différentes informations.

Ces informations peuvent être :

- *Cohérentes :*dans ce cas la solution trouvée par le formateur satisfera toutes les informations données par l'auteur. Dans l'exemple de la Figure III-7, les différentes informations spécifiées sont complémentaires et permettront au formateur de calculer une solution pour le placement temporel des objets.

```
<GroupeTemporel Id="C"
      ListeElement="A B"
      Opérateur="Inside"/>
      <AttributTemporel dur="10 10 10" />
      <ListeRelations >
            <Relation Nom="Before" Objet1="A" Objet2="B">
      </ListeRelations>
</GroupeTemporel>
```

Figure III-7 : Exemple d'informations complémentaires

- *Incohérentes*: dans ce cas le module d'aide du système doit vérifier la cohérence de la spécification en fonction de la sémantique du langage de présentation. Dans l'exemple de la Figure III-8, les durées spécifiées sur l'objet Composite C et l'élément A sont incompatibles avec l'opérateur *Inside*. Dans ce cas, le système essayera de satisfaire la relation, et indiquera à l'auteur qu'il y a une incohérence avec les valeurs des attributs. On peut noter, que si l'auteur avait spécifié un intervalle de valeurs plus large pour l'objet composite, le système aurait de lui-même ajusté cet intervalle de manière à satisfaire l'ensemble de la spécification. Les différentes incohérences possibles seront présentées dans la section 2.4.

```
<GroupeTemporel Id="C" ListeElement="A " Opérateur="Inside"/>
      <AttributTemporel dur="9 9 9" />
      <Element Id="A" />
            <AttributTemporel dur="10 10 10"/>
      </Element>
</GroupeTemporel>
```

Figure III-8 : Exemple d'informations contradictoires

En introduction de cette section nous avons dit que le jeu de relations et d'opérateurs offert par l'environnement idéal est lié au langage de présentation utilisé. Nous allons donner ici le jeu de relations et d'opérateurs défini pour les trois langage SMIL, Madeus et MHML.

Ces relations seront :

- soit une simplification ou une spécialisation des relations proposées par le langage de sauvegarde, et dans ce cas elles seront sauvegardées directement dans le langage de présentation
- soit un raccourci pour manipuler un ensemble de relations temporelles de plus bas niveau. Le mécanisme d'espace de noms offert par XML sera utilisé pour sauvegarder ces informations qui sont propres à l'environnement d'édition.

Ces relations seront définies par le concepteur de l'environnement auteur, mais elles doivent être aussi extensibles par l'auteur s'il en éprouve le besoin. Nous présenterons ce mécanisme dans le chapitre IV section 7.

Nous allons présenter maintenant, le formalisme pour les trois langages qui nous servirons à illustrer ce principe : Madeus, SMIL et MHML.

Dans le cas de langages événementiels, l'environnement fournit à l'auteur la possibilité d'éditer directement les événements du langage par l'intermédiaire d'une palette par exemple.

Le formalisme d'édition pour le langage Madeus :

Le jeu de relations de Madeus étant relativement complet spatialement, nous l'étendons seulement du coté temporel.

Pour cela nous intégrons quatre nouvelles relations aux relations définies dans Madeus-97 (chapitre II section 3.4.2) :

- Le *begin* et le *end*, qui permettent de sortir du cadre des relations d'Allen de base tout en restant dans les relations pointisables (elles peuvent s'exprimer en relation d'instants). Ces relations lèvent la contrainte sur les relations *starts* et *finishes* qui disait que l'objet A devait être plus petit que l'objet B.
- Le *center* et le *sameDuration* : qui nous font sortir des relations pointisables. Ce type de relation impliquera des contraintes sur les mécanismes de formatage que nous utiliserons (chapitre V). La relation *sameDuration* indique que deux objets ont la même durée, sans imposer que ces deux objets commencent en même temps, c'est une généralisation de la relation *equal*. La relation *center*, elle, est une

spécialisation de la relation *during*.

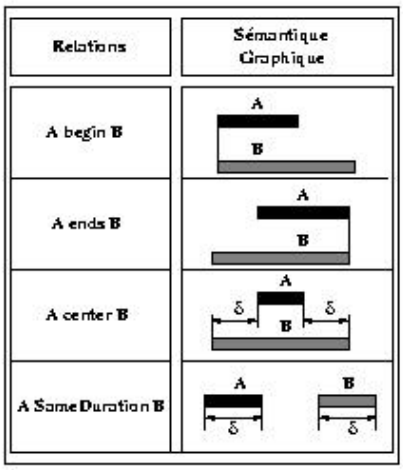


Figure III-9 : L'extension des relations temporelles de Madeus

Le formalisme d'édition pour le langage SMIL :

Le langage SMIL contient essentiellement des opérateurs temporels. Le langage doit donc être essentiellement étendu spatialement. En effet, le langage SMIL permet uniquement de définir un positionnement spatial absolu, il est donc nécessaire pour faciliter l'édition de l'auteur d'offrir une édition de plus haut niveau. Pour cela nous définissons dans le formalisme de l'environnement auteur associé à SMIL les relations spatiales suivantes *: LeftAlign, RightAlign, Center, TopAlign, BottomAlign*. Chacune de ces relations pouvant prendre un paramètre permettant de définir un décalage.

Nous étendons aussi le formalisme d'un point de vue temporel, en insérant l'opérateur *ParEquals* permettant de définir un élément composite de type parallèle avec comme contrainte supplémentaire que tous les fils ont la même durée (voir Figure III-10).

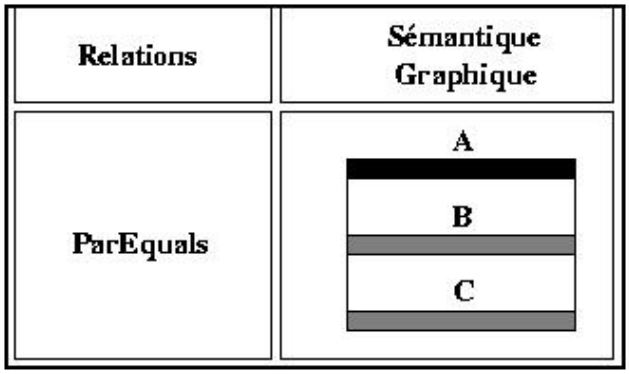


Figure III-10 : L'extension de relation temporelle de SMIL

Le *ParEquals* permet de préciser une sémantique particulière à l'opérateur *Par*, celle-ci sera que tous les objets contenus dans le *Par* auront la même durée.

De plus, dans le cas d'un opérateur *Par*, nous permettons à l'auteur de définir des relations entre les éléments du *Par*. Ces relations sont les relations de Madeus-97 étendues des quatre relations décrites dans la Figure III-9. Cette spécification relative facilitera la tâche lors des phases d'édition en permettant à l'auteur de spécifier le comportement de son document de manière relative.

Le formalisme d'édition pour le langage MHML :

Le langage MHML est de plus bas niveau que les deux langages présentés ci-dessus. Il est donc nécessaire pour faciliter la tâche de l'auteur de définir un jeu de relations de haut niveau plus important que pour les deux langages précédents. Pour cela nous définissons pour les relations temporelles l'ensemble des relations de Madeus-97 (Allen + relations causales) que nous étendons comme pour Madeus des quatre relations décrites dans la Figure III-9.

Nous définissons comme jeu de relations spatiales, les relations spatiales suivantes *: LeftAlign, RightAlign, Center, TopAlign, BottomAlign*. Chacune de ces relations pouvant prendre un paramètre permettant de définir un décalage.

Ce formalisme permet de répondre aux besoins spécifiés dans le chapitre précédent. Les différents besoins en expressivité sont satisfaits (dans le cadre de documents prédictifs avec de la navigation). De plus les besoins de structuration et de support pour la modification du document sont satisfaits grâce d'une part à la structure de groupe et d'autre part à l'utilisation de relations flexibles et la définition des attributs par intervalle.

2.2.3 Les services d'édition de l'environnement idéal

Un système auteur doit fournir un ensemble de fonctions d'édition et de services pour permettre à l'auteur d'éditer son document au travers du formalisme d'édition proposé. Ce formalisme est une extension du formalisme du fichier de sauvegarde.

Dans le chapitre précédent, nous avons listé les différents besoins de l'auteur pendant ce processus, nous allons ici lister les services que le système doit offrir pour répondre en partie à ces besoins. Ces fonctions sont :

- Les opérations de base :
  - l'insertion de nouveaux médias dans le document ;
  - la modification des attributs des médias ;
  - l'ajout de relations spatiales et/ou temporelles.
- Les fonctions plus évoluées : comme le copier/coller, qui peut être un simple copier/coller de médias, mais aussi un copier/coller plus évolué qui permet de dupliquer des groupes d'objets ou les relations entre plusieurs objets.

On peut remarquer que ce sont des fonctionnalités que l'on retrouve dans l'édition de documents structurés [Quint99]. On peut noter, que la réalisation d'une fonction telle que le copier/coller évolué pose de nombreux problèmes. En effet, le copier/coller ne se résume pas seulement à une duplication de la structure et des informations de présentation, mais elle nécessite souvent une transformation de ces informations en fonction du contexte dans lequel on l'insert. Cette transformation ne peut être complètement automatique et doit être assistée de l'auteur ([Bonhomme98]). Dans notre contexte cette difficulté est liée à la notion de groupe ainsi qu'aux relations.

2.2.4 Edition par manipulation directe

Une fonctionnalité offerte par l'environnement auteur idéal doit être l'édition pat manipulation directe. C'est-à-dire que l'auteur doit pouvoir éditer les attributs de ces objets dans les vues spatiale ou temporelle.

De plus, le système doit maintenir les relations existantes entre les objets durant tout le long du cycle d'édition, y compris lorsque l'auteur déplace graphiquement les objets :

- dans la vue de présentation pour les relations spatiales ;
- dans la vue temporelle pour les relations temporelles.

Dans la Figure III-11 l'auteur déplace la bicyclette, et ce dernier reste sur la droite qui modélise une pente. Au cours de ce déplacement le logiciel Cabri [Laborde95] maintiendra les différentes relations que l'auteur a spécifiées.

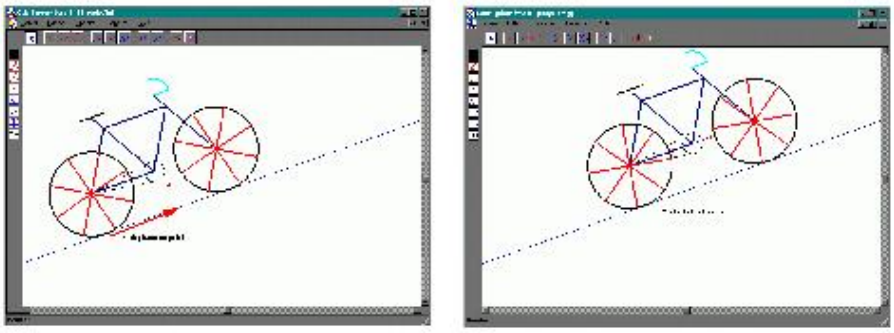


Figure III-11 : Manipulation directe dans Cabri Géomètre II

On peut noter cependant que, dans cette représentation, l'auteur ne visualise pas les relations entre les différents éléments, il en a seulement une perception par les déplacements autorisés. De plus, lors du déplacement, il n'a aucune explication sur le comportement de l'objet, notamment sur les relations qui sont maintenues.

2.3 Spécification d'un système multivues

Nous venons de voir que nous avons choisi d'utiliser un formalisme essentiellement relationnel pour favoriser la tâche d'édition de l'auteur. Cependant, dans le chapitre II section 3.4, nous avons vu que les contreparties à cette flexibilité sont une plus grande complexité de la tâche de visualisation et la nécessité d'automatiser les services de cohérence et de formatage. Nous présenterons les différentes techniques de cohérence et de formatage dans le chapitre V.
Nous allons nous intéresser maintenant aux services de visualisation que doit fournir l'environnement auteur idéal. Ces services de visualisation ne doivent pas être indépendants de l'édition et doivent être liés au processus et aux fonctions d'édition. Ces services de visualisation se feront au travers de *vues*.
Une *vue* est non seulement une entité graphique qui permet de visualiser à l'écran une information, mais c'est une entité qui permet à l'auteur de l'éditer.

2.3.1 Spécification des différentes vues

Étant donnée la complexité des documents multimédias, offrir une seule vue à l'auteur serait insuffisant. Le système auteur doit donc offrir plusieurs vues adaptées à la visualisation des informations attendues par l'auteur. Cependant le système auteur doit aussi fournir des vues pour aider l'auteur, notamment lors de la navigation dans son document. Les vues nécessaires sont :

- La vue de présentation qui permet de présenter le document. Cette vue doit permettre aussi de visualiser les informations spatiales du document. C'est-à-dire que le système offre un mécanisme de visualisation des différentes informations liées aux relations ou aux groupes définis par l'auteur. D'un point de vue édition cette vue doit permettre à l'auteur d'éditer les attributs spatiaux par des manipulations directes.
- La vue temporelle qui présente la dimension temporelle du document et qui permet à l'auteur de naviguer dans l'espace de solutions défini par son document. Cette vue doit aussi offrir des mécanismes permettant de visualiser les relations ainsi que les informations liées à la hiérarchie temporelle. D'un point de vue édition, cette vue doit permettre la manipulation des différents attributs temporels.

- La vue hiérarchique qui visualise les structures temporelle et spatiale du document. Cette vue doit aussi servir de support pour l'édition en facilitant les opérations de restructuration que désire faire l'auteur. Cette vue doit aussi permettre à l'auteur de visualiser les opérateurs associés à chacun des groupes d'objets.
- La vue hypertexte qui permet à l'auteur de visualiser la structure de navigation de son document. Cette vue doit permettre non seulement de voir l'ensemble des liens contenus dans le document, mais elle doit aussi permettre à l'auteur d'avoir une vue plus globale de l'interdépendance entre plusieurs documents.
- La vue objet, qui permet à l'auteur de visualiser l'ensemble des objets présents dans son document.
- La vue attributs, qui permet de visualiser les différents attributs d'un objet du document.
- La vue résumé, qui visualise les instants clés du document. Ces instants clés peuvent être définis par l'auteur ou calculés automatiquement par l'environnement auteur selon plusieurs paramètres (par exemple en ne prenant en compte que les instants de début ou de fin des objets).
- La vue rapport d'exécution, qui permet de visualiser le résultat d'une exécution.
- La vue source, qui permet de visualiser la source du fichier de sauvegarde. Cette vue textuelle peut être une vue qui offre des services de synchronisation évolués (par exemple sur la sélection) entre les éléments de cette vue et les autres vues. L'outil Smartool développé dans l'équipe Oasis de l'INRIA-Sophia Antipolis, offre de telles vues dans le cadre de l'édition de langages dédiés à la programmation.

Nous verrons dans les sections 3 et 4 comment ces vues permettent de répondre aux différents besoins de visualisation et de navigation.

2.3.2 Coopération entre les différentes vues du document

Une des difficultés pour l'auteur vient du nombre de vues nécessaires à la visualisation des différents éléments contenus dans le document. Pour faciliter le travail de l'auteur dans les différentes vues, on peut imaginer une synchronisation sur l'objet sélectionné qui impose au système, lorsque l'auteur change de vue, de focaliser la vue sur cet objet. Ainsi, si l'auteur passe de la vue temporelle à la vue de présentation, cette dernière se positionne à l'instant de début de l'objet sélectionné.

Un autre moyen de faciliter la tâche de l'auteur est de visualiser plusieurs informations dans une vue. Par exemple, la vue temporelle peut non seulement visualiser l'enchaînement temporel des objets mais aussi la structure hiérarchique. La visualisation de cette structure peut permettre à l'auteur de faire plus facilement le lien entre la vue hiérarchique et la vue temporelle. De même, au cours de l'exécution du document dans la vue de présentation, le système peut afficher la progression de la présentation du document. Ces synchronisations sont essentielles pour que l'auteur puisse se repérer facilement entre les différentes vues.

Le fait d'avoir différentes vues permet aussi d'aider l'auteur dans les tâches d'édition complexes, en découpant ces tâches en sous-tâches, chacune s'accomplissant dans la vue la plus adaptée. Nous verrons cela plus précisément dans l'exemple de la section 5.

Aujourd'hui de nombreux systèmes proposent plusieurs vues pour faciliter les tâches de l'auteur. Amaya [Amaya00], par exemple, permet d'utiliser la vue *structure* pour parcourir les pages HTML, ce qui permet à l'auteur d'avoir une vue globale de son document et de naviguer à l'intérieur.

La spécification proposée ici vise à généraliser et à adapter ces principes au contexte multimédia.

Par exemple, pour permettre à l'auteur de faire facilement le lien entre une présentation de son document et la spécification (spécification qui permet de définir un ensemble de présentations), nous offrons en cours d'exécution une ligne temporelle qui se déplace dans la vue temporelle. Cette ligne permet de visualiser l'instant courant de la présentation. Elle permet à l'auteur de faire le lien entre la présentation et le document. Ce mécanisme se retrouve dans Grins et Director.

Notons que l'utilisation d'un système multi-vues nécessite une synchronisation des informations visualisées dans chacune des vues de manière à garantir une certaine cohérence à l'auteur.

2.4 Les services d'aide

Comme nous avons pu le voir précédemment, l'édition de l'auteur peut se faire à deux niveaux : au niveau du formalisme de l'environnement auteur ou alors au niveau du formalisme du langage de sauvegarde utilisé. De ce fait, les services d'aide sont, eux aussi, à deux niveaux.

Le système auteur doit assurer des services complets de vérification de la cohérence, d'aide lors de la spécification (par exemple en l'aidant dans le calcul des durées des objets), mais aussi dans les cas d'erreur ou d'incohérence de documents, en offrant une visualisation et un diagnostique des erreurs sur les deux niveaux possibles de la spécification.

Nous allons maintenant nous intéresser aux différents services d'aide fournis par l'environnement auteur.

La vérification de cohérence : à chaque modification du document, le système d'édition doit vérifier la cohérence du document et calculer une nouvelle solution. En effet, la complexité des documents multimédias est telle que l'auteur ne peut pas appréhender toutes les répercussions possibles d'une modification qu'il a effectuée. Le système doit donc être capable de vérifier en temps réel la cohérence d'un document. Quand une incohérence est détectée le système d'édition doit informer l'auteur de l'erreur, et lui donner un diagnostique aussi complet que possible sur l'erreur de manière à ce que celui-ci puisse rendre le document de nouveau cohérent [Song99]. Il existe différents types de cohérence possibles [Layaïda97, Sabry99] :

- *Cohérence qualitative :* une incohérence qualitative est détectée dans les cas suivant :
  - Si l'auteur met des relations incompatibles entre elles, quelle que soit la valeur des différents attributs sur les objets. Dans la Figure III-12a, on peut voir un exemple où l'auteur a mis 3 relations : A meets B, B meets C, et il essaye de mettre une relation C meets A. Cette dernière relation est incohérente avec les deux autres. Ce type d'incohérence est aussi valable pour les relations spatiales.
  - Si l'auteur définit une relation avec un objet qui n'existe pas.

- Si l'auteur définit des liens temporels incohérents, comme, par exemple, un lien qui pointerait vers la 30ème seconde d'un document qui ne durerait que 25 secondes, ainsi que la détection des objets qui ne seront jamais joués. Par exemple, un objet commençant à la 30ème seconde d'un objet composite qui ne durerait que 20 secondes.

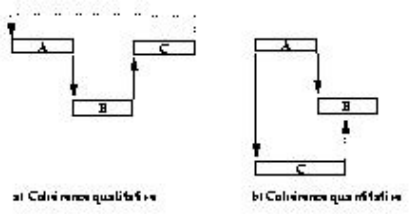


Figure III-12 : Cohérence qualitative et quantitative

- *Cohérence quantitative* **:**une incohérence quantitative arrive lorsque le document est qualitativement cohérent, mais qu'une valeur sur un attribut fait que le document est incohérent. Par exemple, dans Figure III-12b, l'auteur a spécifié que A meets B, A starts C et C finishes A, ce qui est qualitativement cohérent. Mais si comme dans l'exemple A a une durée de 5 unités de temps, B a une durée de 4 et C une durée de 3, les objets A et C ne peuvent terminer en même temps comme l'indiquent les relations. Le document est donc quantitativement incohérent.
- *Cohérence causale*: une incohérence causale est détectée quand un objet doit interrompre un objet qui est déjà fini. Elle est liée à l'insertion de relations telles que : *parmin, parmax, parmaster*. Cette cohérence se ramène dans le cas de documents déterministes à la vérification de la cohérence quantitative [Layaïda97].
- *Cohérence événementielle*: il s'agit de vérifier que les actions associées à l'événement sont réalisables. C'est-à-dire que les objets impliqués dans les actions seront *actifs* (ils seront en cours de présentation au moment de l'activation du lien). Dans le cas particulier des documents prédictifs, on se ramène à la vérification de cohérences quantitative et qualitative. Il faut par exemple vérifier que les objets qui sont la destination des événements soient présents au moment où l'objet source déclanchera l'événement.
- *Cohérence hyper-temporelle***:**la cohérence hyper-temporelle est liée aux liens :
  - Dans le cas de liens globaux, cela ne pose pas de problèmes de cohérence car la navigation hypertexte engendrée par ces liens est orthogonale à l'aspect temporel du document et donc ne crée pas d'incohérence.
  - Dans le cas de liens locaux, il faut vérifier que le lien pointe sur un objet actif. Dans le cadre de documents prédictifs on retombe dans le cas de la cohérence événementielle [Sabry99].

Au cours du processus d'édition, les erreurs sont détectées directement lors de la spécification. Les mécanismes pour détecter ce genre d'erreurs sont relativement simples à mettre en oeuvre. Les principaux algorithmes sont PC2 [Mackworth85], DPC [Meiri92] et DPCincrémental [Chleq95]. Le problème majeur de ces approches est le temps de vérification nécessaire.

Le formatage :

L'environnement auteur idéal doit posséder un formateur. En effet, un besoin essentiel de la phase d'édition est le formatage. La valeur d'un attribut lors de la présentation est le résultat d'un calcul entre les différentes spécification de l'auteur (attribut, relation,). Comme nous avons pu le voir dans les différents langages présentés, il existe différentes méthodes pour spécifier la valeur d'un attribut. La valeur d'un attribut au moment de la présentation peut dépendre de :

- Relations : par exemple si l'on a une relation *alignée à gauche* entre deux objets.
- Intervalle dans lequel il est défini : par exemple si l'attribut X d'un objet est contraint entre les valeurs 50 et100.
- Valeur spécifiée par l'auteur : par exemple si l'auteur a indiqué que l'objet doit se situer si possible en 80.
- Valeur héritée de la structure hiérarchique : par exemple, l'objet composite a l'attribut X qui a pour valeur 60, et tous les fils de cet objet ont un attribut X qui est plus grand que celui de leur ancêtre.

Prenons l'exemple d'un document où l'on possède les informations suivantes :

- Un composite C contenant A et B des objets de type texte (tous les objets contenus dans C sont dans la boîte englobante définie par les attributs X, Y, Hauteur, Largeur de l'objet C).
- L'attribut X de C vaut 30.
- L'attribut X de l'objet A est dans [50..160].
- L'attribut X de l'objet B est dans [130..360].
- L'attribut X de B vaut actuellement 190
- L'attribut X de A vaut actuellement 50

L'auteur ajoute ensuite une relation de la forme :

- A.X = B.X

Le système doit alors calculer une nouvelle valeur pour les attributs X des objets A et B.

Dans le cas ci-dessus, le système doit affecter par exemple la valeur 130 aux deux attributs. Cette valeur satisfait l'ensemble des informations que l'auteur a spécifiées. Sans ce nouveau calcul, le document aurait été quantitativement incohérent.

On peut noter que, lors d'une opération d'édition (la modification de la valeur d'un attribut ou de l'ajout d'une relation), le formateur doit calculer un nouveau placement (spatial ou temporel) pour les objets. Ce calcul doit être rapide lors du processus d'édition.

L'utilisation de méthodes ad hoc de formatage ([Quint98], [Tardif97b]) est aujourd'hui la plus répandue dans les systèmes auteur, cependant l'amélioration des techniques à base de contraintes semble offrir des perspectives intéressantes en termes de simplicité d'expression, d'évolution et de maintenance. Nous étudierons dans le chapitre V ces techniques et nous verrons comment les utiliser dans notre contexte.

Aide au diagnostique : le système doit apporter une aide à l'auteur pour expliquer ces actions dans deux situations :

- Explication lors de la détection d'une incohérence, il doit aider l'auteur dans la compréhension de celle-ci. Cela peut aller de la visualisation des relations impliquées dans l'erreur à la visualisation de toutes les relations induites sur les objets des relations impliquées dans l'erreur.
- Explication du formatage calculé par le système. En effet, le système calculant automatiquement le placement et la durée des objets, ce calcul peut être déroutant pour l'auteur dans certains cas. Il est donc important que le système puisse donner à l'auteur des explications concernant le choix des valeurs pour les différentes variables.

2.5 Synthèse sur la spécification de l'environnement idéal

Nous venons de définir un environnement idéal fournissant un formalisme d'édition, un ensemble de vues pour visualiser et éditer son document au travers du formalisme d'édition offert, et un ensemble de services d'aide tels que la vérification de cohérence, le formatage et le diagnostique pour faciliter la tâche de l'auteur.

Aujourd'hui aucun système auteur n'apporte de réponses pertinentes à ces différents besoins. Au cours des chapitres suivants nous apporterons des réponses concrètes à ces besoins.

L'ensemble des services décrits ci-dessus est complexe du fait qu'il doit à la fois permettre à l'auteur de manipuler facilement le document, tout en essayant de garder un lien avec le langage de présentation. Ce lien doit être conservé pour permettre de satisfaire les besoins liés à la vie du document ainsi qu'au cycle d'édition.

Nous allons maintenant valider ces propositions en indiquant comment elles permettent de répondre aux différents besoins de visualisation (section 3) définis dans le chapitre II, ainsi qu'aux différents besoins de navigation de l'auteur (section 4).

3. Visualisation d'information dans un contexte d'édition

Le système d'édition doit offrir la possibilité à l'auteur d'explorer les informations du document ([Jourdan97b, Jourdan98d]), tout en lui permettant d'adapter l'exploration de l'ensemble de ces informations en fonction de ses besoins (édition, compréhension du comportement de son document,).

Nous allons voir comment l'environnement idéal que nous venons de spécifier permet de répondre aux différents besoins définis dans le chapitre II liés à la fois à la visualisation des médias de base qui constituent le document (section 3.1), mais aussi à la visualisation statique des différentes dimensions qui le composent (section 3.2). Dans la section 4 nous verrons comment les différentes manipulations offertes dans les vues permettront de satisfaire le besoin de perception de l'espace de solutions.

Lors de ces présentations nous essaierons de voir pour chaque besoin quelles techniques permettent de le satisfaire. L'objectif de cette section est donc double :

- vérifier que l'environnement idéal que nous avons spécifié répond aux besoins de visualisation et de navigation définis dans le chapitre II section 2.2.2. Au cours de cette section nous serons amenés à préciser les besoins de visualisation en fonction de chacune des informations traitées.
- étudier quelles sont les solutions techniques existantes pour satisfaire ces besoins. Ceci afin d'évaluer les difficultés liées à la réalisation future de cet environnement idéal.

3.1 Accès aux médias et aux informations de base

Lors des manipulations de médias dans la phase d'édition l'auteur a besoin de visualiser :

- les attributs des médias ;
- la liste des médias manipulables dans son document ;
- la structure des médias complexes (vidéos structurées par exemple).

La visualisation des attributs des médias peut se faire via une palette d'attributs. Cependant, l'auteur a besoin de percevoir non seulement l'aspect quantitatif des attributs mais aussi un aspect qualitatif. De ce fait, visualiser ces informations sous une autre forme est essentiel. Par exemple, les attributs spatiaux seront visualisés dans la vue de présentation.

La visualisation de l'ensemble des éléments d'un document multimédia pose quant à elle des difficultés à cause du facteur d'échelle. L'ordre de grandeur d'un document multimédia est de mille objets pour une présentation d'une dizaine de minutes. L'idée la plus simple consiste à visualiser tous les médias dans une fenêtre comme dans Director en y ajoutant une visualisation des attributs de l'objet sélectionné (Figure III-13). Cette représentation très utile pour de petits documents se révèle inexploitable pour de gros documents. Des techniques à base de filtres peuvent être utilisées pour ne visualiser que les objets d'un certain type ou qui apparaissent dans une certaine scène ou encore les objets non encore utilisés. Ce sont des techniques issues de la visualisation d'informations qui pourraient être utilisées (visualisations en oeil de poisson [Sarkar94] ou sur les murs de perspective [Mackinlay91]). Malheureusement ces techniques ne sont encore que peu employées dans le cadre de l'édition de documents multimédias.

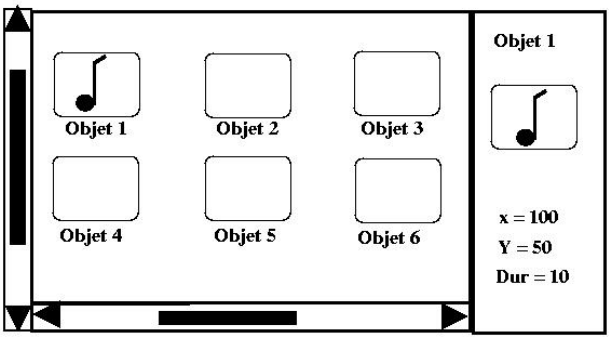


Figure III-13 : Vue des objets présents dans le document

On peut noter que pour des médias comme la vidéo, il est nécessaire de pouvoir visualiser de nombreuses informations. Outre les attributs classiques (spatiaux et temporels), il peut être utile d'avoir des informations sur le nombre de plans, de scènes ainsi que les dépendances temporelles ou spatiales de ces différents éléments.

Dans le domaine de l'édition et l'annotation de vidéo, on peut trouver de nombreux travaux qui permettent d'éditer des parties de vidéos et/ou de les annoter.

Le point commun de tous ces travaux est qu'ils manipulent une structure sur les médias. Parmi ces travaux, on peut citer les travaux de Dumas [Dumas00] qui permettent de définir un modèle d'annotation pour les vidéos, ceux de Chang [Chang97] qui permettent de rechercher de l'information dans des vidéos annotées et enfin ceux de Carrer [Carrer97] qui ont vu la réalisation d'un outil d'annotation. Dans le cadre du projet Opéra les travaux de Tien [Roisin00] visent à intégrer un outil de structuration et d'édition de vidéo au processus d'édition de documents multimédias, permettant ainsi à l'auteur de mettre des relations entre des objets du document et des scènes de la vidéo par exemple. Ces différents travaux n'offrent que peu de représentations graphiques novatrices pour visualiser ces informations et se ramènent souvent à la visualisation de la structure hiérarchique de la vidéo. Cette structure étant synchronisée avec une représentation graphique de la vidéo permettant ainsi à l'auteur de faire facilement le lien entre la structure ou l'annotation et l'image (ou partie de la vidéo) associée.

Dans le cadre de l'environnement idéal que nous avons proposé ces besoins sont couverts par les vues objets, attributs et hiérarchique.

3.2 Accès aux différentes dimensions du document

Dans un premier temps, nous allons nous intéresser à l'aspect visualisation d'informations sans parler des aspects de navigation. Pour chacune des dimensions, nous nous intéresserons dans un premier temps à l'étude de différents travaux permettant de visualiser le type d'informations correspondant sans nous limiter au contexte des documents multimédias, et dans un deuxième temps nous regarderons comment ces techniques sont employées dans notre contexte.

Nous présenterons différents travaux selon le type d'informations visualisé :

- la visualisation d'informations temporelles (section 3.2.1) ;
- la visualisation d'informations spatiales (section 3.2.2) ;
- la visualisation d'informations hiérarchiques (section 3.2.3) ;
- la visualisation d'informations hypertextuelles (section 3.2.4) ;

Nous nous intéresserons aussi à la visualisation des propriétés intrinsèques à notre formalisme d'édition :

- Les informations dynamiques (section 3.2.5) ;
- Les informations sur les relations (section 3.2.6).

3.2.1 Visualisation de l'information temporelle

Un des principaux besoins dans le cadre de l'édition de documents multimédias consiste à visualiser les informations temporelles contenues dans le document.

Ces informations sont de différentes natures :

- attributs temporels ;
- relations et groupes temporels ;
- enchaînement temporel des médias contenus dans le document.

Une solution utilisée classiquement pour représenter de telles informations est une vue temporelle linéaire communément appelée *vue temporelle*. Une telle visualisation est utilisée par exemple dans le cadre d'outils d'édition de vidéo (Adobe première [Adobe-Premiere00]) ou de musique et même dans certains éditeurs de documents multimédias ([Macromédia-Director00], [Jourdan97a]).

Dans le cadre de l'édition de documents à l'aide de relations, une des difficultés est liée à ce que l'auteur manipule un ensemble d'exécutions possibles de son document. Par une représentation statique, il est difficile de faire percevoir à l'auteur l'espace de solutions tout en préservant une réalité temporelle à la représentation. Des métaphores telles que celle des ressorts [Tardif97] peuvent être employées pour indiquer à l'auteur les zones du document qui contiennent de la flexibilité. Dans l'exemple de la Figure III-14 on peut voir un exemple d'une telle représentation.

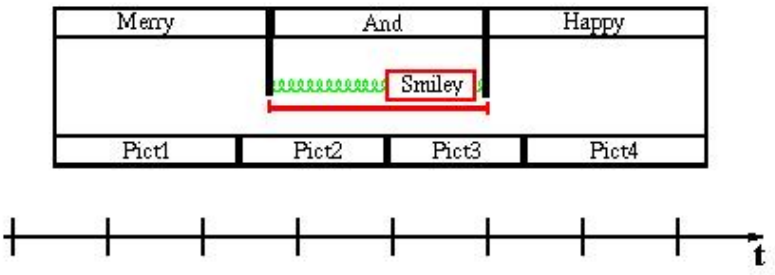


Figure III-14 : Visualisation de la flexibilité dans la vue temporelle

Le principal problème d'une telle représentation est la visualisation de gros volumes d'informations.

On peut noter que dans l'environnement idéal certaines informations sont représentées dans plusieurs vues. Par exemple, les attributs temporels sont visualisés dans les vues attribut et temporelle ce qui permet à l'auteur d'utiliser la vue la plus adaptée à son besoin. Par exemple, en phase de création d'un média la vue attribut lui permettra de positionner tous les attributs sur son média, la vue temporelle, lui permettant elle, d'ajuster le placement temporel entre les médias. De la même manière la structure de groupe temporel est visualisée dans les vues hiérarchique et temporelle.

Les problématiques de visualisation de relations étant communes à toutes les informations, nous les présenterons ultérieurement à la fin de cette section dans le cas de la visualisation des relations. Les techniques d'affichage de gros volumes d'information seront elles présentées dans la section visualisation de données hiérarchiques.

Dans la section 4 nous verrons comment des techniques de navigation peuvent améliorer la perception de cet espace de solutions.

3.2.2 Visualisation d'informations spatiales

Les informations spatiales contenues dans le document sont de différentes natures :

- attributs spatiaux ;
- relations et groupes spatiaux ;
- placement spatial des médias dans le temps.

Les méthodes de visualisation du placement spatial offertes dans l'environnement idéal pour les objets respectent le principe du WYSIWYG utilisé dans le cadre des éditeurs graphiques ainsi que pour l'édition de documents classiques. L'environnement auteur permet, par exemple, à l'auteur d'afficher certains instants précis du document dans la vue de présentation. Dans la Figure III-15 nous pouvons voir que l'auteur a stoppé la présentation de son document à un instant précis.

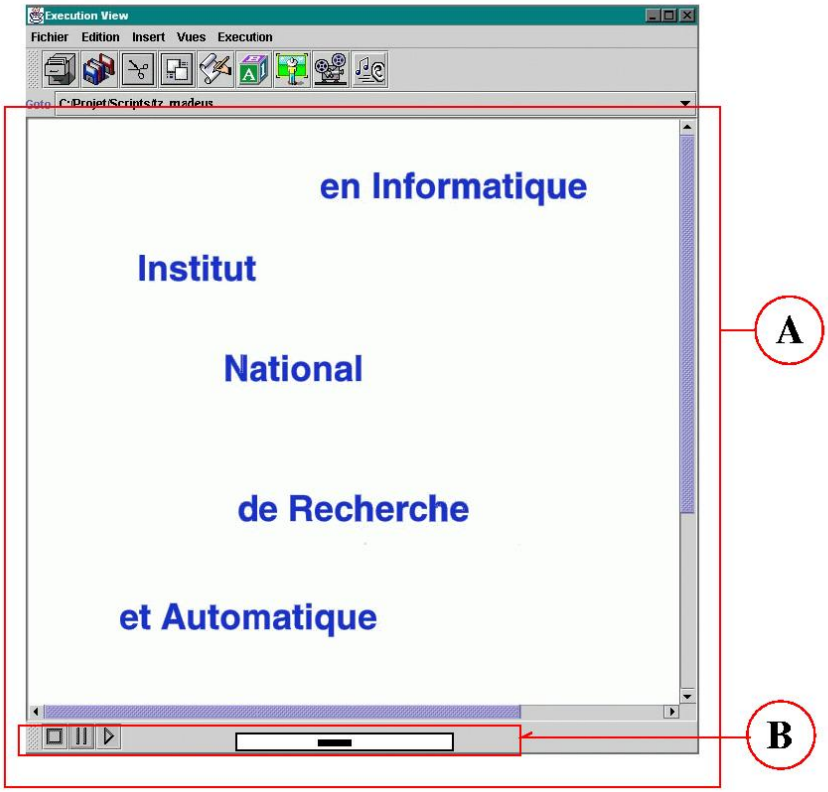


Figure III-15 : Vue de présentation

La limite de cette approche est que l'on ne peut pas voir comment deux objets qui n'ont pas d'instant de présentation en commun sont placés l'un par rapport à l'autre. Il faut donc pouvoir présenter plusieurs instants en même temps.

Dans Grins (Figure III-16), une vue permet de visualiser en même temps toutes les régions du document. On a ainsi sur la même vue, la

représentation spatiale d'objets ou de zones qui ne sont pas présents temporellement aux mêmes instants.

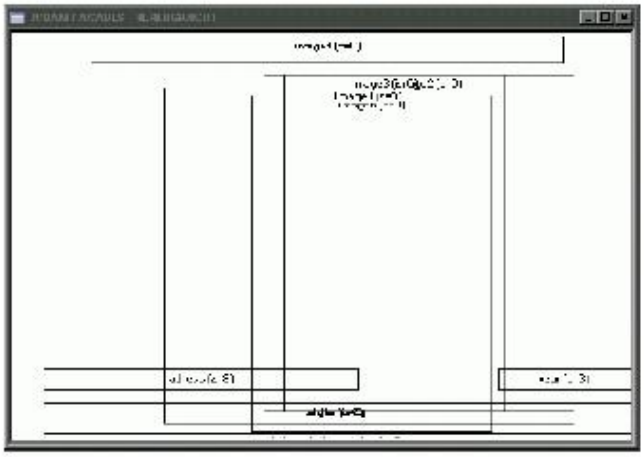


Figure III-16 : Visualisation des régions dans Grins

Ce principe peut être affiné de manière à pouvoir par exemple visualiser toutes les régions utilisées sur une période de temps, ou toutes les régions d'un sous-ensemble d'objets du document. Supposons que nous ayons un document composé de trois objets, chacun associé à une région (Figure III-17), si nous visualisons simultanément les 3 régions à l'écran nous obtenons la visualisation de toutes les régions du document. Cet affichage, fort utile pour des présentations de type transparent (pour permettre le placement d'objets comme les titres, aux mêmes endroits) se révèle vite inutilisable pour d'autres types de documents du fait du nombre de régions présentes dans le document. Pour remédier à cela des techniques de filtrage peuvent être appliquées lors de la visualisation pour par exemple ne visualiser que les régions utilisées pendant les 30 premières secondes, Il n'est malgré tout pas toujours facile de trouver un bon filtre, celui-ci étant dépendant de l'opération qu'est en train de faire l'auteur.

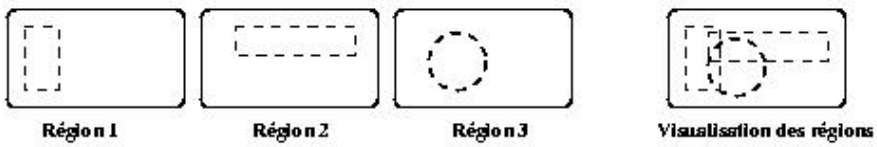
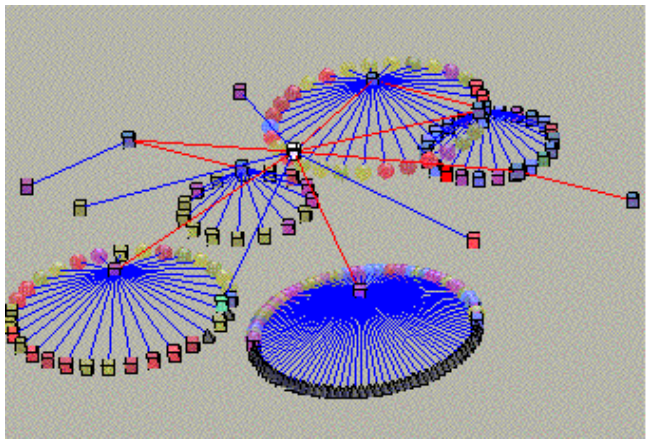


Figure III-17 : Visualisation simultanée des régions

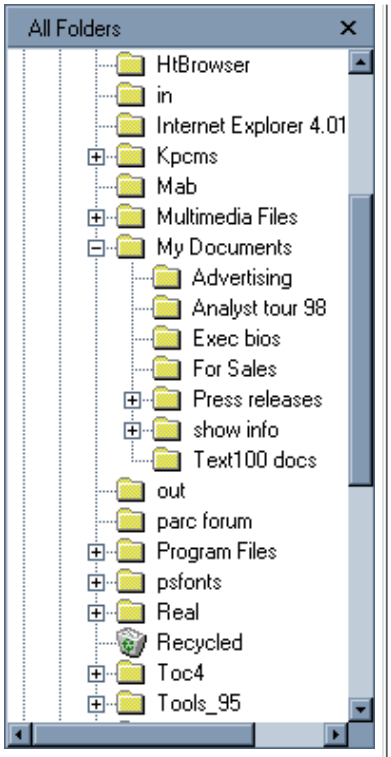
3.2.3 Visualisation d'informations hiérarchiques

La visualisation d'informations hiérarchiques est utile pour visualiser les structures hiérarchiques (spatiale, temporelle) du document.

Le problème que l'on rencontre communément dans ce contexte est celui du facteur d'échelle. Des techniques telles que l'utilisation de filtres (filtres sur le type des noeuds, sur la profondeur, sur la valeur de certains attributs) peuvent permettre d'alléger l'affichage. De plus, des techniques graphiques peuvent permettre d'améliorer la lisibilité de l'affichage. De telles techniques ont été développées essentiellement dans le cadre de la visualisation de systèmes de fichiers. Les techniques les plus connues sont à base d'arbres (Figure III-18a) ou d'arbre coniques (Figure III-18b, [Robertson91]). Dans le cadre de l'édition de documents, ce sont les techniques à base d'arbres qui sont le plus communément utilisées. On peut par exemple citer Amaya [Amaya00] qui utilise une telle représentation pour visualiser la structure XML du document (Figure III-19).



b) Représentation à base d'arbres coniques



a) Représentation à base d'arbres

Figure III-18 : Représentations hiérarchiques d'un système de fichier

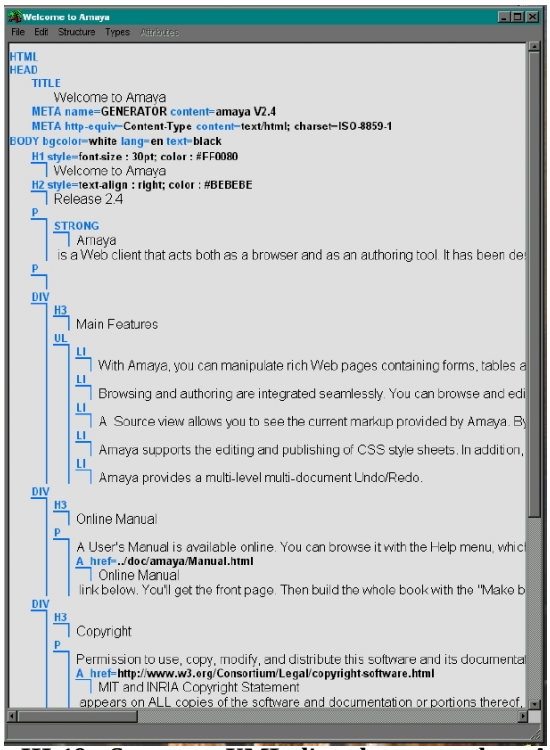


Figure III-19 : Structure XML d'un document dans Amaya

3.2.4 Visualisation d'informations sur la structure hypertexte

La dernière composante à laquelle nous allons nous intéresser est la visualisation de la structure hypertexte.

Les besoins dans le cadre de l'édition de documents multimédias sont similaires à ceux liés à l'édition hypertexte :

- visualisation des liens de navigations ;
- visualisation des liens locaux ;
- visualisation de l'interdépendance entre deux ou plusieurs documents ;
- visualisation de l'espace de navigation du lecteur.

Avec l'avènement du Web, l'explosion des pages HTML et la complexité de plus en plus croissante des sites Web, de nombreux travaux ont essayé de faciliter la perception globale des sites.

On peut noter que les besoins dans le cadre de documents multimédias recouvrent ceux qui sont liés à la dimension hypertexte mais la dimension temporelle soulève un nouveau problème. Il est important de visualiser la partie de document qui n'a pas été jouée à cause de l'activation du lien. Dans le cas de documents prédictifs, on a la connaissance de cette partie, et il est important de permettre à l'auteur de la visualiser. Ce besoin que l'on avait déjà dans la navigation hypertexte classique n'était pas satisfaisable dans ce contexte du fait que l'on n'avait pas la connaissance réelle de ce qu'avait vu (ou lu) le lecteur.

Dans la Figure III-20 on peut visualiser un exemple d'affichage de la structure d'un document hypertexte. Comme le montre cette figure, cette représentation sans doute très utile pour de petits documents devient vite inexploitable pour des documents complexes.

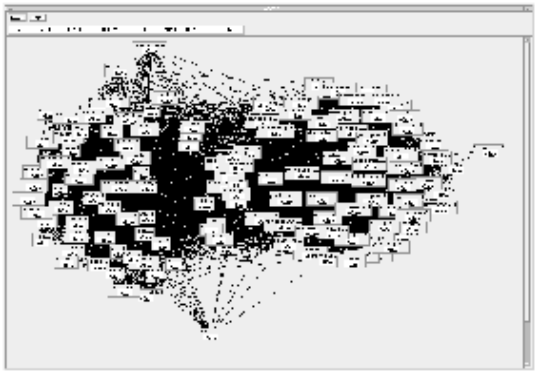


Figure III-20 : Visualisation de la structure hypertexte d'un document

Des techniques à base de filtres (sur le type des liens, ou sur le type de document, la localité des documents) doivent être utilisées pour alléger l'affichage.

Dans le cadre de l'édition de documents, peu de systèmes intègrent de tels outils. La métaphore la plus communément utilisée est la visualisation du lien par une icône sur le média. Les systèmes n'offrent pas une vue globale de la structure.

Dans le cadre de notre environnement idéal, la vue hypertexte doit permettre de visualiser ces différentes informations (liens de navigation, interdépendance entre documents).

Aucun système ne propose une visualisation de l'espace précis de navigation du lecteur. Ce besoin, difficilement satisfaisable dans le cadre de document hypertexte classique du fait que l'on ne connaît pas ce que le lecteur a réellement lu d'un document, peut être satisfait dans le cadre des présentations temporisées.

La *vue rapport d'exécution*, permet à l'auteur de visualiser une exécution réelle de son document, et de ce fait, elle peut être utilisée pour visualiser l'espace de navigation accédé au cours de la présentation par le lecteur.

3.2.5 Visualisationde l'exécution du document

Le système auteur idéal doit permettre à l'auteur de visualiser les différentes informations liées à l'exécution de son document. Ces informations sont :

- Visualisation d'une exécution a priori : le système auteur au travers de la vue temporelle offre une visualisation a priori de l'enchaînement temporel du document. Le choix de la solution présentée peut-être paramétré pour choisir par exemple quels liens le lecteur activera, et à quel moment se fera leur activation.
- Visualisation d'une exécution et donc de la dynamicité du document : la vue de présentation du document permet à l'auteur de visualiser une exécution. Il est opportun d'offrir à l'auteur des fonctions telles que :
  - *Joue, qui permet à l'auteur de lancer l'exécution du document à partir du début.*
  - *Joue_à(instant) démarre l'exécution à un instant donné.*
  - *Joue_à(objet) lance l'exécution du document à partir de l'objet spécifié*

Ces trois fonctions sont très utiles lors de la phase d'édition car elles permettent à l'auteur de visualiser le document selon une localité temporelle.

- Visualisation du résultat d'une exécution : la vue rapport d'exécution permet à l'auteur de visualiser le résultat réel d'une exécution (activation de liens). Se pose alors le problème de l'édition dans cette vue rapport d'exécution. Editer le document au travers de cette vue peut paraître plus simple pour l'auteur, par exemple pour corriger un comportement non désiré. Cependant cela pose le problème de la pertinence de cette édition vis-à-vis du document qui, lui, représente un ensemble d'exécutions. En effet, quelles garanties l'auteur peut-il avoir sur sa modification vis-à-vis de tous les scénarios possibles ?

3.2.6 Visualisation des relations contenues dans le document

Une des difficultés rencontrées dans les différentes dimensions est de visualiser les relations que l'auteur a spécifiées entre les différents objets et leurs conséquences.

Cette visualisation peut se faire à différents niveaux. On peut, par exemple, enrichir chacune des vues pour afficher cette information, c'est le cas de la visualisation des arcs de synchronisation dans la vue temporelle de Grins. On peut aussi visualiser explicitement le graphe de dépendance, comme c'est réalisé dans le logiciel FireFly [Buchanan93] (Figure III-21). Dans cette figure, on voit que le système représente explicitement les instants de début et de fin des objets, les synchronisations étant représentées par des liens entre les noeuds. L'inconvénient majeur de ce genre de représentation est qu'il ne donne pas une représentation temporelle de l'enchaînement des événements.

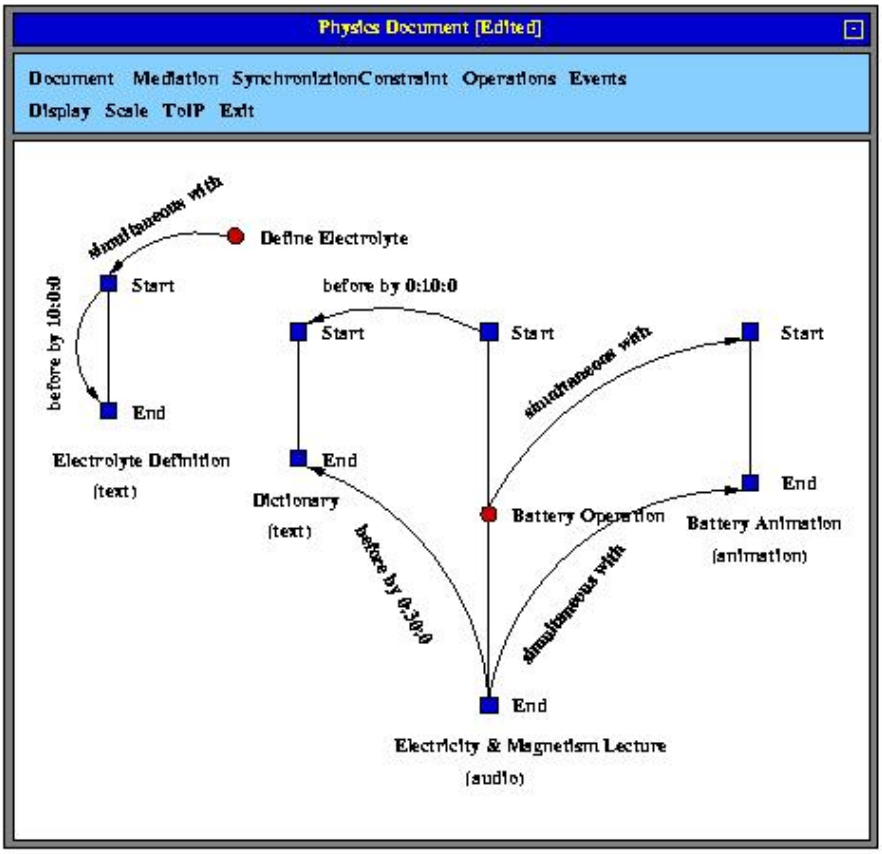


Figure III-21 : Graphe de synchronisation de Firefly

Parmi les outils qui offrent une visualisation de relations dans des domaines autres que l'édition multimédia, on peut citer Action Works d'IBM (Figure III-22) qui permet à l'auteur de visualiser de manière abstraite l'ensemble des relations qui composent un Workflow. Un workflow est la description d'un ensemble de tâches qui sont liées les unes par rapports aux autres temporellement. Nous donnerons une définition plus précise dans le chapitre VI section 3.

Sur la Figure III-22 on peut voir que différentes métaphores graphiques sont utilisées en fonction du type de relations que l'on visualise. Le système représente les différentes possibilités par des icônes différentes (rendez-vous, condition, enchaînement automatique, enchaînement en cas d'erreur). Cette représentation permet facilement de voir les dépendances entre les tâches, cependant elle ne permet pas de percevoir l'enchaînement temporel de ces dernières.

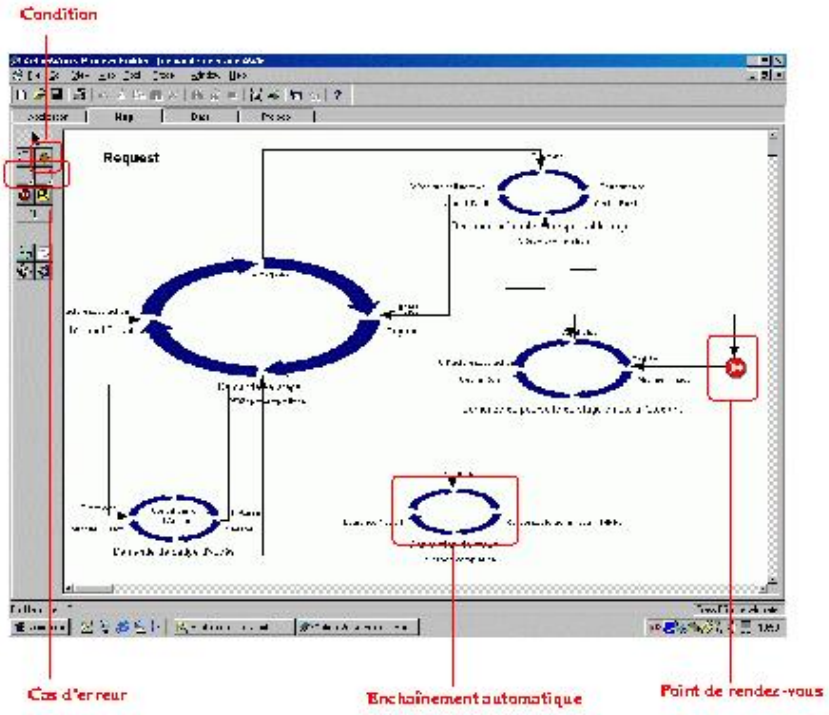


Figure III-22 : Vue d'édition d'Action Works Project Builder

Le système doit aussi fournir des mécanismes de synchronisation forte entre chacune des vues et la liste des relations. Par exemple, lorsque l'on

sélectionne un objet dans la vue spatiale, les relations impliquées dans son placement doivent être aussi visualisées.

On retrouve ce type de fonction dans le logiciel Cabri [Laborde95]. Cabri permet entre autres de dessiner des figures géométriques en spécifiant des relations entre les objets. Ainsi, l'extension réalisée par Bellynck [Bellynck99] permet de synchroniser la liste des relations avec les objets manipulés.

4. Parcours dans le document multimédia

Dans de nombreux domaines (hypertexte, recherche d'informations) des métaphores graphiques (boîtes, ressorts, arbres coniques) sont utilisées pour permettre à l'auteur de mieux appréhender les gros volumes d'information comme c'est le cas de certains documents multimédias. Cependant ces métaphores graphiques ne sont pas suffisantes ([Morse99], [Wilcox97]). Il est important de compléter ces métaphores graphiques statiques par des moyens de parcours à l'intérieur du volume d'informations offert à l'auteur. La barre de défilement en est un exemple simpliste, mais d'autres mécanismes peuvent être proposés comme les arbres hyperboliques. La représentation graphique choisie permet d'afficher à l'auteur l'information qui est essentielle à son travail en cours, tout en lui permettant d'accéder aux autres informations par des manipulations simples.

Nous allons nous intéresser dans cette partie à des techniques de navigation développées dans l'environnement idéal de manière à aider l'utilisateur dans sa tâche de compréhension d'un gros volume d'information.

4.1 Type de parcours offerts dans l'environnement idéal

Des techniques ont été développées pour permettre à l'auteur de naviguer de façon plus rapide à l'intérieur de son document. On peut classer ces techniques en trois catégories :

- Navigation basée sur la structure du document : c'est, par exemple, les types de navigation que l'on retrouve dans Thot et Word qui permettent de naviguer de titre en titre, de figure en figure, ou, plus généralement, sur le type ou sur la valeur d'un des attributs des noeuds du document.
- Navigation basée sur la temporalité du document : on peut citer les fonctions d'avance rapide, de retour, de pause/resume qui permettent à l'auteur de naviguer temporellement dans le document. Ces fonctions peuvent être mises en oeuvre dans la vue de présentation mais aussi dans la vue temporelle.

Les modes de parcours peuvent également tirer parti des services de synchronisation entre les vues. On peut par exemple imaginer une synchronisation sur les objets en cours de visualisation entre ces vues de navigation et les autres vues de l'application. Cela pourrait se faire au moyen d'une métaphore graphique qui, par exemple, dans la vue hiérarchique se traduirait par le changement de couleur des objets en cours de visualisation.

On peut citer aussi les techniques de navigation plus évoluées qui se basent sur les instants clés du document (images clés représentant les instants de début ou de fin d'objets (voir Figure III-23) ([Layaida99])), les techniques qui permettent de compresser certaines parties temporelles. On peut noter que ces techniques sont largement utilisées dans les outils de création et de manipulation de vidéo [Adobe-premiere00].

Figure III-23 : Visualisation des instants clés

4.2 Parcours de l'espace de solutions du document

Nous avons vu au cours du chapitre précédent que lors de l'édition de documents à base de relations flexibles, l'auteur ne spécifie pas une présentation de son document mais un ensemble de présentations. Le système doit donc permettre à l'auteur de naviguer dans cet espace. Un moyen simple de réaliser cela est de permettre à l'auteur de manipuler directement la flexibilité des relations [Tardif97]. Un besoin important lors de ces opérations est la perception de l'espace des déplacements possibles des objets avant toute manipulation. Par exemple, lorsque l'auteur sélectionne un objet dans le but de le déplacer dans la dimension spatiale ou temporelle, il est important que le système l'informe de l'intervalle dans lequel il pourra effectuer ce déplacement sans enfreindre les différentes relations du document qu'il a déjà spécifiées. Le système auteur doit donc dans un premier temps calculer l'intervalle de déplacement possible de l'auteur et dans un deuxième temps le rendre perceptible à ce dernier, par exemple en l'affichant [Tardif97].

Cet intervalle n'est pas seulement dû aux propriétés intrinsèques de l'objet, mais aussi à son contexte dans le document et notamment aux différentes relations directes ou indirectes qui le lient aux autres objets.
Par exemple dans la Figure III-24, si l'auteur définit le document suivant :

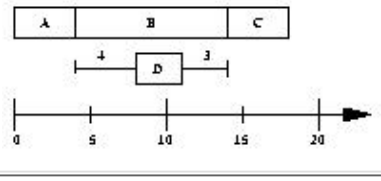
Objets	Relations	
A: [4] B: [10] C: [4] D: [1 3 6]	A Suivi_de B B Suivi_de C A Avant D D Avant C	

Figure III-24 : Besoin d'anticipation en cours d'édition

La solution courante est affichée dans la Figure III-24, avec A=4, B=10, C=4, D=3. Maintenant supposons que l'auteur déplace l'objet D. Il est important de l'informer que ce déplacement peut se faire uniquement de quatre unités vers la gauche ou de trois unités vers la droite (par une borne de déplacement comme sur la figure, par exemple).

4.3 Synthèse sur le parcours des informations contenues dans le document

Les différentes vues spécifiées dans la section 2.3 répondent aux besoins de visualisation définis dans le chapitre précédent (section 2.2.2), en permettant à l'auteur de comprendre et de manipuler les différents aspects de son document. De plus, certaines vues, comme la vue temporelle et spatiale, permettent à l'auteur d'éditer simplement le document, par exemple, en plaçant directement les objets à leur emplacement temporel ou spatial.

Le parcours des informations contenues dans le document au travers des vues permet à l'auteur, entre autre, de visualiser l'espace de solutions défini dans son document et lui permet de parcourir, selon ses besoins, les différentes parties du document. De plus, la combinaison des vues et des fonctions de navigation permet à l'auteur d'avoir à sa disposition plusieurs manières pour effectuer chacune des opérations d'édition. Il peut alors choisir celle qui convient le mieux à l'opération qu'il fait ou la plus adaptée à la succession d'opérations qu'il est en train de faire. Nous illustrerons cela dans l'exemple de session d'édition dans la section qui suit.

5 Illustration au travers d'un exemple

De manière à illustrer ce chapitre et le type d'environnement auteur que nous proposons, nous allons terminer ce chapitre par un exemple de session d'édition avec un tel environnement.
Nous allons utiliser l'environnement auteur idéal construit au dessus de Madeus. Ce choix, est purement arbitraire, l'édition se faisant de manière similaire dans tous les environnements auteur.
Nous allons donc reprendre le document INRIA présenté dans le chapitre précédent.
Dans un premier temps, l'auteur au moyen de l'interface va créer ces cinq médias texte (`"Institut", "National", "de Recherche ", "en Informatique", "et Automatique"`).
Dans l'exemple de la Figure III-25, on peut voir cette création. L'auteur a plusieurs possibilités pour créer ces objets. Par exemple, il peut insérer l'objet directement dans la vue spatiale ou hiérarchique (flèches 1 de la figure), et ensuite positionner les attributs soit par manipulations directes dans les vues (attributs spatiaux dans la vue de présentation et attributs temporels dans la vue temporelle) ou grâce à la vue attributs (flèches 2 de la figure).

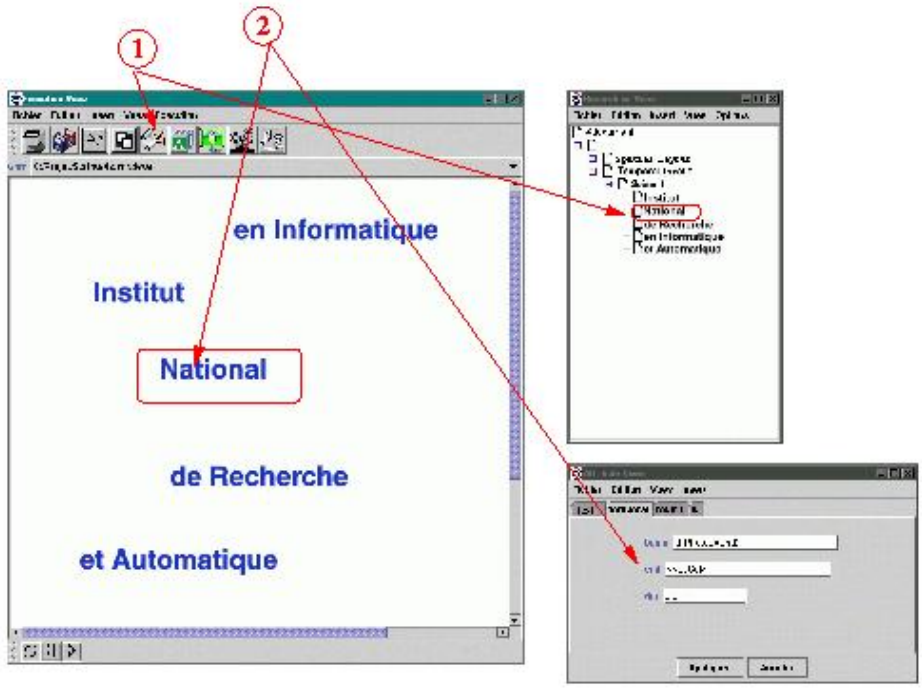


Figure III-25 : Ajout d'un élément texte

Dans un deuxième temps, il va définir la structure spatiale et créer un composite qui contient ces cinq médias. Il va ensuite mettre quatre relations pour aligner ces objets horizontalement (`<AligneAGauche param1="media1" param2="media2" delta="+60">`), et quatre relations pour aligner ces objets verticalement (`<Under param1="media1" param2="media2" delta="+20">`). Dans l'exemple de la Figure III-26, l'auteur insert une relation spatiale entre les objets "National" et "et Automatique". Pour cela il peut sélectionner les objets dans la vue qu'il préfère (dans l'exemple, la sélection

s'effectue à l'aide des vues spatiale et hiérarchique), et il positionne une relation à l'aide de la palette de relations.

Une fois le positionnement spatial défini, il peut définir la structure temporelle en définissant un composite qui contient les cinq médias et placer les objets les uns par rapport aux autres avec des relations spatiales et/ou temporelles (exemple : `<start param1="media1" param2="media2" delay="10s"> ).`

Dans l'exemple de la Figure III-27, l'auteur insert une relation temporelle entre deux objets. Comme pour les relations spatiales il peut sélectionner les objets dans la vue qu'il désire, et positionne une relation grâce à la palette temporelle.

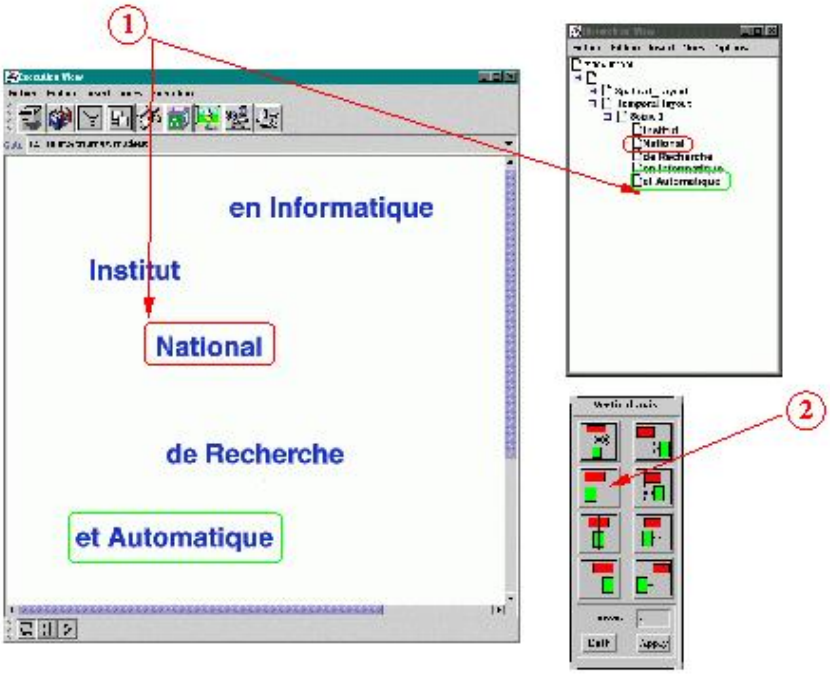


Figure III-26 : ajout de relation spatiale

Une fois ces définitions faites, il peut placer spatialement le média qui apparaît en premier là où il le désire dans la vue présentation. Du fait que le système auteur maintient les relations, les autres objets seront toujours placés en cascade les uns par rapport aux autres. Il fait de même pour le premier objet temporel.

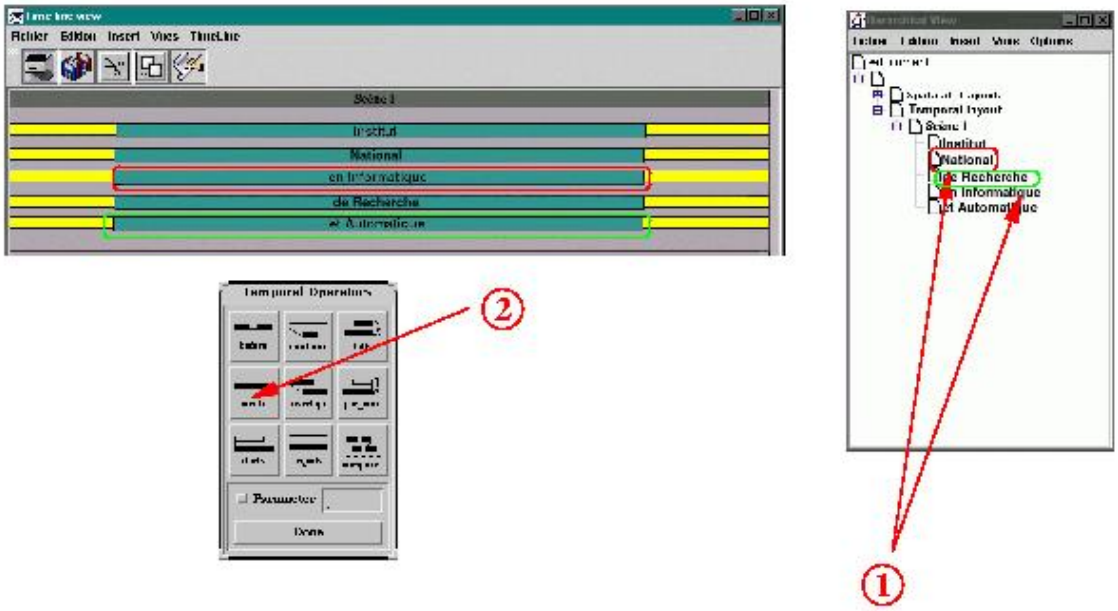


Figure III-27 : Ajout de relation temporelle

A tout moment au cours de ce processus d'édition l'auteur peut exécuter son document de manière à avoir un aperçu de son document.

On peut noter que s'il désire rajouter une image de titre, il lui suffit de créer le média, de le placer spatialement et temporellement, et si nécessaire de mettre une relation avec un des autres objets pour que la présentation s'adapte automatiquement. De la même façon, s'il désire que sa présentation fasse 10 secondes de plus, il lui suffit de spécifier cette durée sur le document, et le formateur modifiera les durées sur les objets pour prendre en compte cette modification.

On peut aussi imaginer que l'auteur veuille utiliser cette présentation pour réaliser un autre document, par exemple la présentation de la boîte à outils Kaomi. Pour cela, il remplace le contenu des cinq médias texte par ("Kaomi :", "une boîte", "à outils", "pour la construction",

"d'environnements auteur". Le système ajustera alors le placement spatial des cinq objets en conservant les relations spatiales spécifiées.

6 Bilan et perspectives de l'environnement idéal

Au cours de ce chapitre nous venons de proposer un formalisme d'édition et un environnement auteur pour la spécification de documents multimédias. Dans l'exemple terminant ce chapitre, nous avons pu voir qu'il était relativement simple de définir le document, et qu'il était relativement simple de le modifier.

L'hypothèse que nous avons faite, qui était de fournir un environnement auteur basé essentiellement sur un formalisme d'édition relationnelle (tout en permettant à l'auteur d'utiliser le formalisme du fichier de sauvegarde), répond dans un premier temps aux différents besoins énoncés dans le chapitre II (section 2.1 et 2.2).

Les différentes propositions qui ont été faites dans ce chapitre permettent aussi de répondre aux différents besoins de visualisation et de présentation définis dans le chapitre II.

L'environnement résultant permet donc d'apporter à la fois une réponse sur la simplicité d'édition ainsi que sur les services d'aide à l'auteur tel que la visualisation, la vérification de cohérence et le formatage. Ce sont généralement ces points qui limitent l'utilisation des outils actuels.

Nous allons nous intéresser maintenant à la réalisation d'un tel environnement. L'environnement que nous allons proposer ne satisfait pas tous les points décrits dans ce chapitre. Cette limitation porte essentiellement sur les trois points suivants :

- visualisation de la structure hypertexte ;
- visualisation de gros volumes d'information ;
- aide au diagnostique.

Pour ces trois points de nombreux travaux sont en cours et ils ne sont pas liés à la dimension temporelle des documents multimédias.

Nous apporterons par contre une réponse à tous les autres points. La volonté de valider les principes d'édition pour de nombreux langages représentatifs des différentes approches de spécifications a motivé le choix de réaliser une boîte à outils de manière à mettre en commun le maximum de compétence. Les points difficiles dans la réalisation de cette boîte à outils sont :

- la mise en oeuvre du formalisme d'édition ;
- la réalisation des différentes vues ;
- la gestion du placement temporel et spatial des éléments ;
- la vérification de la cohérence du document.

La présentation des réponses à ces problèmes se fera en trois temps. Nous présenterons dans un premier temps l'architecture de la boîte à outils ainsi que la mise en oeuvre du formalisme d'édition et des différentes vues dans le chapitre IV.

Dans un deuxième temps (chapitre V), nous présenterons la réalisation des deux derniers points, que nous avons choisi de réaliser à l'aide de technologies à base de contraintes. Le choix d'utiliser des techniques à base de contraintes est motivé par le fait que nous voulons apporter une réponse globale aux problèmes de placement et de cohérence pour les différents langages de documents multimédias que nous allons expérimenter.

Dans un troisième temps (Chapitre VI) nous présenterons la réalisation de différents environnements auteur de documents multimédias.

Chapitre IV : Kaomi

1. Kaomi, une boîte à outils

Lors de la présentation de l'environnement idéal, nous avons vu (Chapitre III section 2.2.2) qu'il existait une dépendance entre le langage de présentation et le formalisme d'édition proposé par l'environnement. Cependant au cours du chapitre III nous avons identifié un noyau commun, que nous avons appelé formalisme d'édition de l'environnement auteur, ainsi que l'ensemble des fonctions d'édition qui s'y rattachent. Il nous a donc semblé naturel de réaliser une boîte à outils pour factoriser le coeur de l'édition de documents multimédias (structures de données, fonction d'édition) et ainsi nous permettre d'expérimenter plus facilement plusieurs environnements auteur construits sur des formats de sauvegarde différents.
Une des difficultés se situe dans la conception de l'architecture de cette boîte à outils car de celle-ci dépend la facilité avec laquelle nous serons capable de créer des environnements auteur. De plus, étant développée dans un contexte de recherche cette boîte à outils doit être facilement modifiable et extensible pour servir de support à différentes expérimentations.
Nous allons dans un premier temps décrire l'architecture générale d'un environnement auteur construit avec Kaomi (section 2). Nous présenterons ensuite le principe de Kaomi et les différents services offerts par celle-ci (section 3) avant de présenter comment ces services seront utilisés lors de la réalisation d'un environnement auteur (section 4). Une fois que nous aurons présenté le principe général de Kaomi et de son utilisation nous nous intéresserons plus particulièrement à son implémentation (section 5 à 9) avant de présenter les différents environnements auteur réalisés (chapitre VI).

2. Architecture générale

Dans la Figure IV-1, on peut visualiser l'organisation globale de l'architecture d'un environnement auteur réalisé avec Kaomi. Kaomi a été écrite en Java, et utilise un certain nombre de bibliothèques écrites en C (Ansi) telles que certains résolveurs de contraintes ou en Java (JMF, Swing).
L'utilisation du langage Java a permis une structuration du code grâce à l'approche objet, et une portabilité des environnements auteur écrits grâce à la machine virtuelle Java.

Figure IV-1 : Structure générale d'une application utilisant Kaomi

Le concepteur d'environnement auteur qui utilise la boîte à outils profite donc de sa portabilité pour créer des environnements auteur multi plates-formes.

3. Principes de Kaomi

La boîte à outils Kaomi fournit un ensemble de services pour le concepteur d'environnement auteur de documents multimédias. Ces services sont de plusieurs niveaux (Figure IV-2) :

- Services de chargement : des facilités pour gérer des fichiers XML (section 3.1).
- Services d'édition de documents (voir section 3.2) :
  - Un ensemble de structures de données pour manipuler et éditer des documents multimédias.
  - Un ensemble de services d'édition pour aider l'auteur dans sa tâche.
- Service de gestion de vues : qui permet de gérer la coopération et la cohérence entre les différentes vues. Cet ensemble de vues permet de visualiser le document, naviguer à l'intérieur des différentes informations contenues dans le document mais aussi d'éditer le document (voir section 3.3).

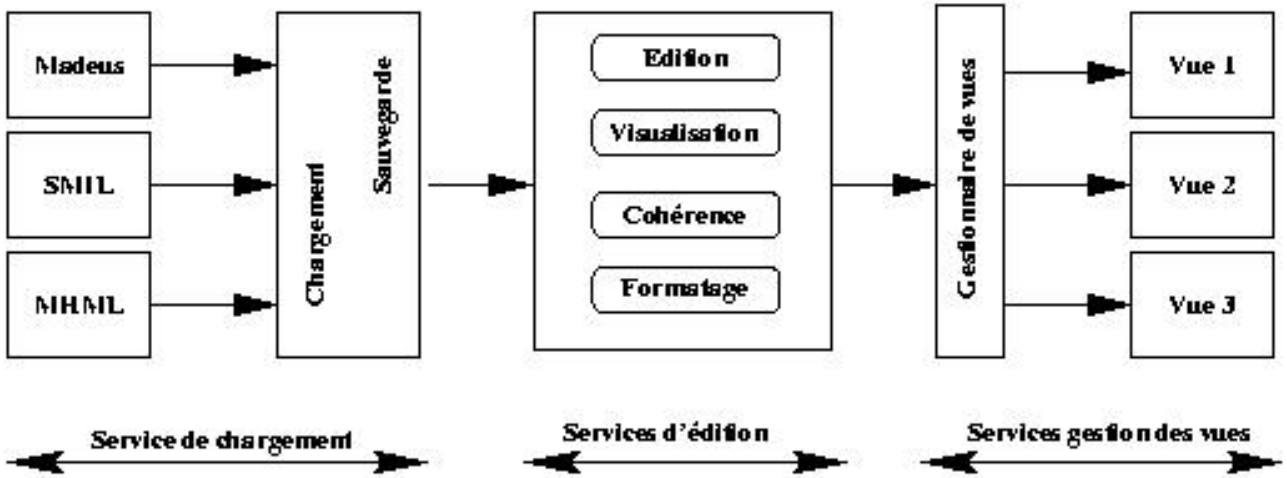


Figure IV-2 : Structures et services fournis par Kaomi

3.1 Service de chargement/ sauvegarde

L'hypothèse faite dans Kaomi est que les fichiers source des environnements auteur construits seront décrits sous une forme textuelle respectant la syntaxe XML. Ce choix est motivé par le fait qu'XML est le standard le plus adapté pour la définition de documents et qu'aujourd'hui la plupart des formats de documents multimédias non propriétaires utilisent cette syntaxe.
Le service de chargement de Kaomi se base donc sur la structure XML du fichier source pour fournir un ensemble de services, comme les analyses lexicale et syntaxique. Ces services sont construits au-dessus du parseur Xerces d'Apache[Apache-Xerces00] et permettent de vérifier qu'un fichier est conforme à la syntaxe de balises d'XML (on dit alors que le document est bien formé) et qu'il respecte la grammaire (DTD) du langage qu'il utilise (on dit alors que le document est valide).
Kaomi offre différentes fonctionnalités pour permettre au programmeur d'inclure un ensemble d'actions, pour notamment construire ses propres structures de données au cours de ces analyses.
Le service de chargement permet par exemple au développeur d'un environnement auteur d'inclure un ensemble de fonctions spécifiques lors du traitement d'un élément XML.
Le service de sauvegarde permet de sauvegarder à la fois les informations liées au format de sauvegarde mais aussi celles spécifiques de l'environnement auteur via le mécanisme d'espace de noms fourni par XML.

3.2 Services d'édition

Kaomi fournit un ensemble de services d'édition pour tous les environnements construits à partir de cette boîte à outils. Ces services d'édition sont basés sur une structure de données appelée *format pivot de Kaomi* qui sera présentée dans la section 6. Ces services sont :

- Modification des attributs sur les objets.

- Ajout / retrait de médias.
- Ajout / retrait de relations spatiales ou temporelles.

Lors de la modification du document, Kaomi maintient la cohérence du document. Kaomi ne permettra pas au document d'entrer dans un état incohérent. Si la modification de l'auteur est incohérente, elle est annulée par le système, dans le cas contraire le système formate une nouvelle solution pour prendre en compte la modification et présenter le document.

Dans chacun de ces cas, la boîte à outils calcule de manière efficace la (ou les) nouvelle(s) solution(s).

Kaomi fournit aussi un ensemble de services pour aider les auteurs quel que soit le langage dans lequel ils spécifient leur document. Ces services sont :

- Formatage, pour aider l'auteur dans la tâche fastidieuse de spécifier les instants de début et de fin de tous les objets, ainsi que pour le calcul des durées. Ce service pourra être spécialisé en fonction du langage cible de l'environnement auteur.
- Détection des incohérences, ce service permet à l'auteur de spécifier des documents toujours cohérents, quelles que soient les relations qu'il a spécifiées entre les objets et les valeurs temporelles qu'il a affectées aux médias.
- Visualisation des relations, ce service permet d'afficher en surimpression dans la vue temporelle ou dans la vue d'exécution les relations et donc les interdépendances entre les objets.

L'ensemble des mécanismes pour la détection de cohérence et le formatage sera présenté dans le chapitre V.

3.3 Service de gestion de vues

Kaomi fournit un ensemble de vues qui est basé sur celui de l'environnement idéal (Chapitre III section 2.3). De manière à faciliter la coopération entre les différentes vues, Kaomi fournit aussi un gestionnaire de vues qui s'occupera de cette coopération. Ce gestionnaire sera présenté dans la section 9.

Les différentes vues fournies par Kaomi sont :

- La vue de présentation qui visualise l'exécution du document. Cette exécution pourra être synchronisée avec la vue temporelle de manière à voir la progression de l'exécution par exemple. Cette vue servira de plus à visualiser l'ensemble des informations spatiales.
- La vue temporelle qui permet d'afficher et de manipuler les informations temporelles. Cette vue servira aussi de support à la visualisation du rapport d'exécution.
- La vue hiérarchique qui visualise les structures spatiale et temporelle du document.
- La vue textuelle qui affiche le fichier source du document.
- La vue résumé qui permet de naviguer dans un résumé du document, ou dans les instants clés du document.
- La vue attributs qui visualise les attributs de l'objet sectionné dans le document. Nous avons défini une vue plutôt qu'une simple palette du fait que nous voulons une synchronisation entre les valeurs affichées dans cette vue et les différentes vues du document.
- La vue relation qui visualise sous forme textuelle la liste des relations d'un objet.

- La vue rapport d'exécution et de navigation qui permet à l'auteur de visualiser en phase de conception les différents comportements de son document lors d'exécutions.
- La vue vidéo structurée qui permet d'éditer de façon plus fine la structure de la vidéo en permettant par exemple de définir des scènes à l'intérieur de la vidéo.

L'ensemble de ces vues, ainsi que leurs mécanismes de construction et de synchronisation, seront présentés dans la section 9. La vue vidéo structurée qui est l'objet d'une thèse en cours sera, elle présentée dans le chapitre VI.

4. Utilisation de Kaomi

4.1 Principe d'utilisation de Kaomi

Kaomi n'est pas seulement une boîte à outils classique qui fournit un ensemble de services utilisables par le concepteur d'un environnement auteur. En effet, Kaomi n'est pas seulement une boîte noire fournissant un ensemble de services, mais elle est plus proche d'un logiciel extensible et adaptable à plusieurs niveaux grâce à une approche objet. Le concepteur peut ainsi intégrer de nouveaux services ou de nouvelles vues. Dans cette perspective, nous avons prévu une utilisation à plusieurs niveaux (Figure IV-3). Ces niveaux vont de l'interface graphique perceptible par l'auteur, au coeur de la boîte à outils et de ses structures de données :

- **Services de chargement**: nous avons fait l'hypothèse de manipuler des fichiers XML. Le concepteur d'un environnement d'édition peut insérer un ensemble d'actions lors du chargement du fichier pour construire ses propres structures de données et/ou utiliser celles de Kaomi.
- **Services d'édition**: l'utilisateur peut soit utiliser une des fonctionnalités prévues au sein de Kaomi, soit l'étendre, soit la remplacer, soit en rajouter une. Nous présenterons ces services dans la section 8.
- **Services de gestion des vues**: il existe deux possibilités d'extension au niveau des vues. La première s'effectue grâce au gestionnaire de vues qui permet d'intégrer statiquement une nouvelle vue sans remise en cause du code des vues existantes. La deuxième méthode est d'étendre une vue existante en lui ajoutant, par exemple, de nouvelles fonctionnalités.

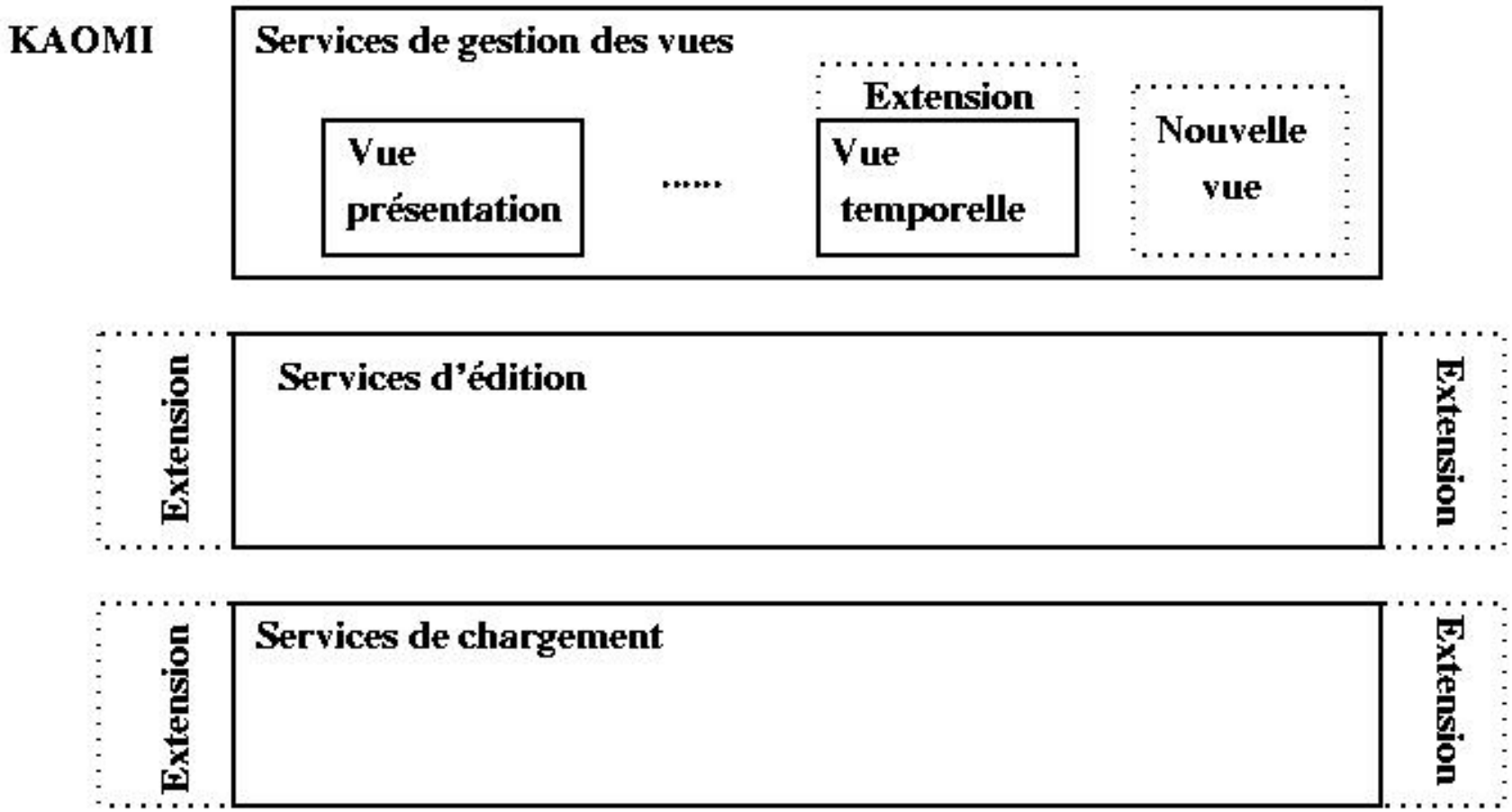


Figure IV-3 : Principe d'utilisation de Kaomi

Ces différents services reposent sur une structure de données appelée format pivot de Kaomi (section 6). Il existe là aussi deux manières d'étendre cette structure de document, la première est d'ajouter des attributs spécifiques sur les noeuds du document, la deuxième est d'étendre les noeuds du document avec de nouvelles fonctionnalités (fonction d'édition, de manipulation) ou de modifier celles existantes.

4.2 Les environnements réalisés avec Kaomi

Aujourd'hui la boîte à outils Kaomi est utilisée dans des outils auteur représentatifs des différents formalismes de spécification de documents multimédias. Nous allons donner ici une liste de ces outils auteur, nous présenterons plus précisément leur implémentation dans le chapitre VI après la présentation complète de Kaomi dans les chapitres IV et V.

Les outils auteur de documents multimédias réalisés avec Kaomi :

- Madeus Editeur : un outil auteur du langage Madeus ;
- SMIL Editeur : un outil auteur du langage SMIL ;
- MHML Editeur : un outil auteur du langage MHML.

Ces outils auteur couvrent deux formalismes de spécification que nous avons présentés dans le chapitre II.

Nous illustrerons, au cours de ce chapitre, les différents aspects liés à l'utilisation de Kaomi au travers de ces différents outils auteur.

5 Implémentation de Kaomi

Maintenant que nous avons vu les différents services de Kaomi, nous allons nous intéresser à sa conception et plus particulièrement au mode de programmation qui a permis une telle extensibilité. Nous présenterons, dans la section 5.1, les mécanismes de base nécessaires à l'implémentation de Kaomi. Dans la section 5.2, nous présenterons l'architecture de cette boîte à outils. Nous présenterons ensuite plus particulièrement le format pivot de Kaomi sur lequel repose en partie les fonctionnalités d'édition (section 6), et plus précisément celles liées à la dimension temporelle (section 7). Dans les sections 8 et 9, nous décrirons plus complètement les services d'édition fournis ainsi que les différentes vues offertes par Kaomi. Enfin, nous ferons une comparaison avec d'autres formats de documents et d'autres services d'édition fournis par des boîtes à outils d'édition (section 10).

5.1 Description des mécanismes de base de Kaomi

L'utilisation de Kaomi repose en partie sur l'utilisation de services fournis par le langage Java. Dans la boîte à outils nous utilisons principalement trois fonctionnalités de Java pour l'extension :

- La programmation objet et les mécanismes d'héritage (section 5.1.1).
- Le mécanisme d'interface de Java (section 5.1.2).
- Le mécanisme de ressource (section 5.1.3).

5.1.1 Programmation objet et héritage

L'héritage est un des mécanismes que nous utilisons pour étendre les fonctionnalités de la boîte à outils. Le développeur d'une application au-dessus de la boîte à outils peut facilement étendre une classe, en créant une sous-classe, et en écrivant les différentes fonctionnalités qu'il désire. Ce type d'extension est favorisé et simplifié par l'utilisation des mécanismes de ressources et d'interfaçage qui permettent par exemple, au programmeur de rajouter facilement un accès pour l'auteur vers une nouvelle fonction, au travers d'un menu de l'environnement.

5.1.2 Présentation du mécanisme d'interface Java

Le mécanisme d'interfaçage de Java, que l'on peut rapprocher de ceux offerts par ADA, C++, se distingue de ces derniers par l'aspect dynamique de la résolution des dépendances, aspect dynamique que l'on retrouve dans des langages comme Guide [Balter94].

Une interface Java est un fichier dans lequel on décrit les prototypes d'un ensemble de fonctions.

Deux utilisations des interfaces sont possibles :

- Une classe peut *implémenter* une interface, cela signifie que cette classe fournie une implémentation de toutes les méthodes décrites dans l'interface.
- Une classe peut manipuler des variables dont le type est une interface. Cela signifie que l'instance de la variable manipulée implémente les fonctions de l'interface.

Ce mécanisme permet de faire abstraction, au moment de l'écriture d'une classe, des classes manipulées au travers d'une interface. Ce mécanisme permet aussi de partager facilement un service entre plusieurs classes.

C'est cette utilisation que nous allons présenter plus précisément.

Dans Kaomi nous avons défini une interface *Tree*. Cette interface décrit un ensemble de fonctions nécessaires pour manipuler des arbres (voir Figure IV-4).

```
public interface Tree {
	Tree getParent() ;
	Tree[] GetChildren() ;
	boolean isLeaf();
	boolean isNode();
}
```

Figure IV-4 : Exemple d'interface, l'interface Tree

La classe VueHiérarchique manipule des objets de type *Tree* et les affiche sous une forme arborescente.

La classe Document de Kaomi implémente l'interface *Tree*. Les classes qui héritent de la classe document (DocumentVueHiérarchique et DocumentVueTemporelle) implémentent donc cette classe.

Ces deux classes peuvent donc être visualisées dans la vue hiérarchique.

La classe DocumentVueTemporelle implémente en plus de l'interface *Tree* l'interface InterfaceDocumentVueTemporelle ce qui lui permet d'être aussi visualisée dans la vue temporelle.

Dans la Figure IV-5 on peut voir la représentation de ce mécanisme.

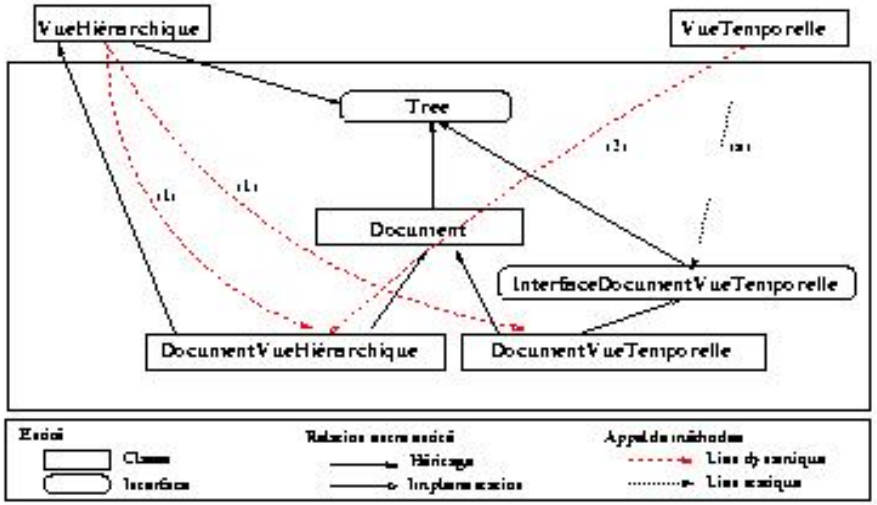


Figure IV-5 : Exemple d'utilisation du mécanisme d'interface de Kaomi

5.1.3 Le système des ressources de Java

Le système de ressources est basé sur deux éléments :

- Des fichiers de ressources : ils permettent de définir des ressources sous forme textuelle. Toute information nécessaire à l'application peut être décrite sous forme de ressources : label des menus de l'interface graphique, messages d'erreur, répertoires par défaut, Le système de ressources permet de définir intrinsèquement des ressources configurables pour différentes langues.
- Un système de chargement et d'accès aux ressources. Java fournit tous les mécanismes nécessaires pour accéder aux fichiers de ressources dégageant ainsi le programmeur du chargement de ces informations. Le programmeur demande une ressource, et c'est le système qui s'occupera de charger le fichier nécessaire et d'accéder à la valeur de la ressource.

Dans Kaomi, nous utilisons ce principe pour toutes les données que nous chargeons dynamiquement. Par exemple, ce mécanisme est utilisé pour construire les menus de l'application. Dans la Figure IV-6 on peut voir une partie de la description du menu *Insert*. Le concepteur décrit que ce menu est composé de trois éléments : Image, Vidéo, Texte. Pour chacun de ces éléments il décrit le label qui lui est associé, ainsi que l'action qui sera appelée lorsque l'auteur sélectionnera ce menu.

```
//ElémentGraphique.NomDeLaVue.Actions = listes des actions possibles
Menu.executionView.Insert = Image Video Texte

// label associé à l'action insert image dans la vue d'exécution
Menu.executionView.Insert. Image.label = Image

//commande associée à l'action insert image dans la vue d'exécution
Menu.executionView.Insert. Image.action =Insert Image
```

Figure IV-6 : Exemple de ressources : les menus

Une autre utilisation des ressources est la mise à jour de la liste des classes fournissant un service. Par exemple dans le cas du module contrainte, nous avons un ensemble de résolveurs à disposition, le fichier de ressources permet de modifier la liste des résolveurs disponibles sans modifier le code de l'application.

Dans l'exemple de la Figure IV-7, on indique au système qu'il existe actuellement trois classes disponibles qui implémentent l'interface *solver*.

Le système pourra ainsi choisir dynamiquement en fonction de critères la classe qu'il désire utiliser. Ce mécanisme sera décrit dans le chapitre V.

```
//liste des résolveurs
Solver = PC2 cassowary jSolver

//classes implémentant les résolveurs
Solver.PC2 = FR.inria.opera.kaomi.solver.PC2
Solver.cassowary = EDU.washington.solver.cassowary
Solver.jSolver = JP.jSolver
```

Figure IV-7 : Exemple de ressources : les résolveurs

Le mécanisme de ressource est aussi utilisé pour configurer l'environnement de l'auteur et ses préférences (taille des fenêtres, fenêtres ouvertes par défaut,).

5.2 Structure de classes

Nous allons maintenant nous intéresser plus particulièrement à l'organisation de cette boîte à outils. Nous allons décrire les principales classes de Kaomi, classes dont nous pouvons voir l'organisation sur la Figure IV-8.

Cette structure a pour objectif de permettre la réalisation de l'environnement auteur idéal. Dans ce but, nous avons défini un ensemble de classes

permettant d'offrir une gestion des services communs (*KaomiManager*), de gérer l'édition simultanée de plusieurs documents (*DocumentManager*) et d'éditer un document au travers d'un ensemble de vues (*ViewManager,WindowsManager, ReferenceDocument*).

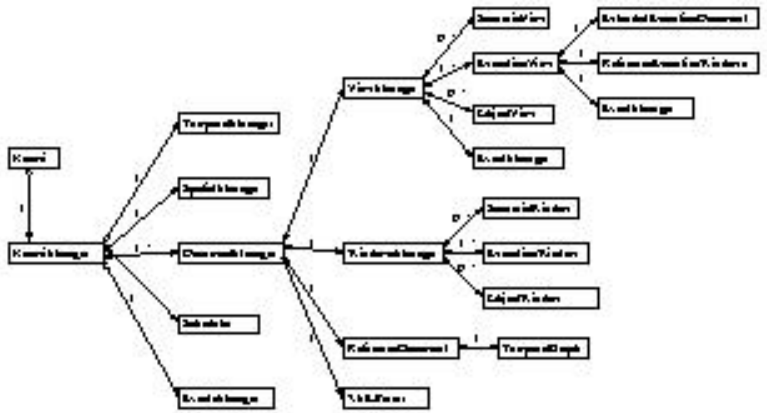


Figure IV-8 : Structure des classes de Kaomi

Kaomi : la classe Kaomi gère le contexte de l'application. Par contexte, on désigne, les préférences utilisateur ou l'accès aux différentes ressources.

Les services :

La classe *KaomiManager* permet de gérer un ensemble de documents et l'accès aux services partagés, parmi ceux-ci :

- Le gestionnaire de présentation ou *scheduler* qui à partir d'un graphe temporel présente le document.
- *Les gestionnaires temporel et spatial:*ils sont accessibles *du document de référence* mais aussi des différentes *vues*. Par exemple, la vue temporelle les utilise pour calculer le placement spatial des objets temporels.
  - *TemporalManager* : c'est une classe qui permet de gérer les informations temporelles ainsi que les services associés (formatage, cohérence,).
  - *SpatialManager* : c'est une classe qui permet de gérer les informations spatiales ainsi que les services associés.
- *DocumentManager* : cette classe permet l'édition synchrone dans plusieurs vues d'un document.
- *Les analyseurs lexicaux et syntaxiques XML*: ce service fournit les mécanismes nécessaires à la lecture de fichier XML, il est spécialisé en fonction du document manipulé et permet de construire la structure de données. De ce fait, ce module sera associé au gestionnaire de document.

Cette structuration a pour but de favoriser l'extension des différents services. Par exemple, les services de formatage ne sont pas intégrés avec le document. Cela permet de ne pas faire dépendre le formatage du document d'un résolveur précis. Nous verrons de manière plus précise comment se passe le formatage dans le chapitre V.

L'édition synchrone dans plusieurs vues :

La classe *DocumentManager* gère un document lors d'un processus d'édition et de visualisation. Cette classe gère donc le fait qu'un document est visualisé dans un ensemble de vues, et initialise la mise en place des différents mécanismes de synchronisation entre les différents éléments qu'elles contiennent. C'est cette classe qui permet de maintenir la cohérence entre la structure de données de *document de référence* et les copies de document présentes dans les différentes vues.

Un *document manager* contient :

- *ReferenceDocument* : c'est une des classes au coeur de l'application. C'est elle qui offre une structure de données générale pour décrire et stocker les informations contenues dans les documents multimédias. Les opérations d'édition s'effectueront sur cette structure. Cette structure de données sera construite lors de la phase de lecture du fichier de sauvegarde.
- *Window*: cette classe implémente tous les services de gestion d'une fenêtre graphique : création, gestion des menus et des actions associées aux menus, gestion des événements. Ces différents services seront appelés par les vues.
- *WindowsManager* : c'est une classe qui permet de gérer les différentes fenêtres physiques d'un document. Elle s'occupe de créer les objets graphiques et s'assure qu'une fenêtre graphique est associée à une vue.
- *ViewManager* : c'est une classe qui permet de gérer l'ensemble des vues d'un document et notamment la synchronisation entre ces vues. De plus, il s'assure de la cohérence entre toutes les vues, notamment lors des opérations de sélection et de navigation.

Mécanisme de vues :

Ce mécanisme repose sur la classe ViewManager qui contient les classes:

- *Vue :*Chacune des vues de Kaomi utilise un ensemble de services communs à toutes les vues, et possède en plus une structure de données qui lui est propre, une copie transformée du document de référence (voir section 9), une fenêtre graphique et un gestionnaire d'événements. L'entité vue s'occupe de maintenir la cohérence entre la copie du document qu'elle contient et l'affichage dans la vue qu'elle gère.
- *ExtendedDocument*: c'est une classe qui d'un point de vue fonctionnalité est proche de la classe référence document. Elle offre un service supplémentaire, celui d'être synchronisable avec le document de référence. C'est à dire qu'elle sera notifiée de toute modification réalisée sur le document de référence.
- *Référence vers une window*: lors de la création d'une vue, le gestionnaire de fenêtres affecte une fenêtre à la vue. Cette fenêtre sera utilisée pour afficher le document étendu. La distinction entre vue et fenêtre a été réalisée de manière à mettre en commun le maximum de services. En effet, les fenêtres de toutes les vues sont relativement proches et offrent des mécanismes similaires qui ont été mis en commun.

Dans la Figure IV-9, nous présentons le mécanisme général de création des vues. Ce mécanisme sera décrit précisément dans la section 9.

A la lecture du document source la structure de données qui servira de format pivot à Kaomi sera construite (ReferenceDocument). Les différentes vues et leurs documents étendus seront construits par transformation (filtrage, ajout d'informations complémentaires) du document de référence. Cette transformation est réalisée de manière automatique par Kaomi. Cependant, elle est spécialisable pour chaque environnement auteur construit avec Kaomi.

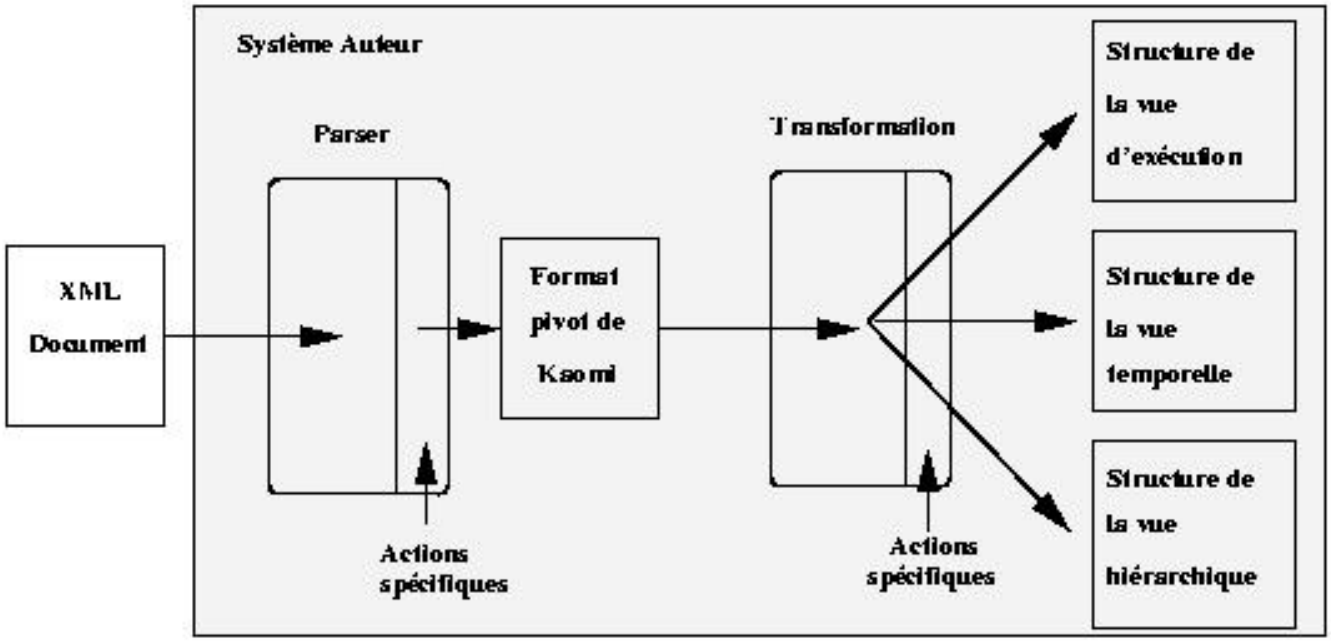


Figure IV-9 : Mécanisme de création des vues

La classe ReferenceDocument ainsi que les classes documents dans les vues héritent d'une classe Document. Nous présenterons plus complètement cette classe dans la section 6.

6 Le format Pivot de Kaomi

Un des objectifs de Kaomi est de fournir un environnement auteur pour de nombreux langages multimédias du fait du nombre croissant de langages (voir Chapitre II). De manière à éviter l'écriture d'une structure pour chaque format, il a fallu concevoir des classes qui offraient un formalisme suffisamment général pour permettre à une grande famille de langages de s'intégrer dedans. Un des autres intérêts d'avoir une structure de données commune pour plusieurs langages est de permettre la mise en commun d'un ensemble de services d'édition.

L'objectif de ce format pivot est donc triple. Il doit permettre l'édition, la présentation du document, mais aussi faciliter la création des structures de document dans les différentes vues.

Dans ce but nous avons défini une structure de document hiérarchique composée de trois entités :

- L'entité *média*, qui est l'entité de base de l'application (section 6.1).
- L'entité *élément multimédia* qui est une entité de plus haut niveau qui nous permet d'utiliser des objets élémentaires en faisant abstraction du type de média qu'ils représentent, mais aussi de définir les informations (temporelles et spatiales) liées à l'insertion des médias dans le

document(section 6.2).

- L'entité *document*, qui permet de représenter l'information nécessaire à l'édition d'un document multimédia. Elle permet notament de définir le regroupement des éléments média pour définir leur enchaînements temporels et spatiaux dans le document (section 6.3).

Nous allons maintenant présenter ces trois entités.

6.1 Les médias dans Kaomi

Dans Kaomi, les différents éléments de base que l'on peut utiliser dans un document sont les médias image, vidéo, texte, son, HTML, Applet.

Kaomi utilise les JMF pour accéder et présenter les médias : image (JPEG, GIF), vidéo (AVI, MPEG, MOV) et audio (MP3, WAV).

Par rapport aux services fournis pas les JMF, nous avons ajouté deux médias de base de plus : les objets HTML et les Applets. Cette intégration a été facilitée dans le cas de HTML par l'utilisation de la bibliothèque (graphique) intégrée dans Java (Swing) et dans le cas des Applets par l'utilisation de Java qui nous fournit intrinsèquement une machine virtuelle pour les Applets.

Un ensemble d'interfaces décrit les attributs pour chaque type de média. Cet ensemble peut être étendu pour intégrer de nouveaux médias. Cette structure est basée sur le modèle proposé par Sabry dans le cadre du gestionnaire de présentation de documents multimédias [Sabry98].

6.2 Les éléments multimédias de Kaomi

Un objectif de la structure de données est de permettre la définition de fonctions d'édition simples (ajouter un fils, supprimer un fils, ajouter un attribut, mettre à jour un attribut (voir section 3.2)) et la définition d'une aide pour l'écriture des services de chargement et de sauvegarde.

Dans cette perspective, la structure de données pour représenter les documents multimédias est basée essentiellement sur un élément multimédia qui contient les informations temporelles, spatiales, de navigation (hyperliens), le type ainsi que les informations concernant les médias (objets son, image).

Dans la Figure IV-10, on peut voir une représentation graphique de cette structure de données. Les attributs de navigation, de type et ceux spécifiques au langage sont stockés comme attributs de l'élément multimédia.

Ces éléments sont les feuilles de la hiérarchie du document. Ils permettent de faire abstraction au cours des différentes opérations d'édition du type du média manipulé. Par exemple, cela permettra de faciliter la création de la palette d'attributs du fait que les attributs associés aux médias seront homogènes.

Cette structure permet aussi d'être indépendant du média, l'auteur peut ainsi changer le média associé à un élément multimédia, sans avoir à changer les synchronisations avec les autres objets multimédias. De plus, un même média peut être utilisé par plusieurs éléments multimédias.

Frank Duluc [Duluc00b] a proposé et validé le même niveau d'abstraction dans un contexte de gestion d'un fond documentaire multimédia.

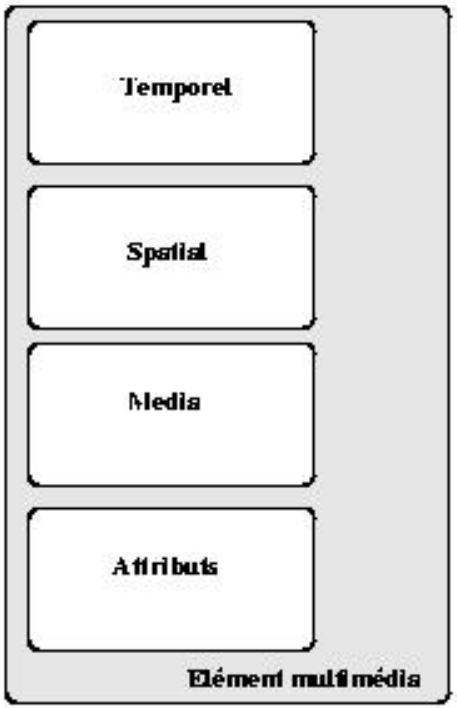


Figure IV-10 : Format pivot de Kaomi

Cet élément multimédia s'intègre dans une structure de document plus complète. Nous allons maintenant présenter cette structure.

6.3 La structure de document de Kaomi

Il existe aujourd'hui plusieurs modèles pour représenter des documents structurés. Le modèle le plus connu est DOM (Document Object Model [DOM98], [DOM2-00]) que nous allons présenter maintenant car il sert de base à notre structure de document. Il existe des extensions de DOM plus particulièrement adaptées au monde du document multimédia comme SMIL-DOM par exemple [SMIL-DOM00] que nous présenterons ensuite.

DOM : est une interface faite pour créer et manipuler des documents structurés. L'intérêt d'une telle structure est de fournir une interface qui peut être largement utilisée pour différents types d'environnement et d'application. DOM est issu de DHTML, mais il est beaucoup plus général que ce dernier qui ne fait référence qu'à HTML. Pour définir une telle structure, l'interface de DOM décrit une hiérarchie de noeuds. Cette hiérarchie est composée d'une entité *document* et de *noeuds*. L'entité *document*, racine de l'arbre, est une entité qui représente le document et un ensemble d'informations attachées à un document (titre, fichier, auteur, date de création,). Les noeuds sont décorés par des informations, certaines sont prédéfinies d'autres non. Cela permet par exemple à toutes les applications de stocker leurs données dans la structure.

L'interface DOM fournit aussi un ensemble de méthodes permettant de manipuler cette structure.

L'intérêt d'utiliser une structure comme DOM est de s'appuyer sur une structure de données standardisée et donc d'un ensemble de fonctions générales définies pour sur cette structure.

SMIL-DOM : est une spécialisation de DOM pour les documents SMIL. Cette interface, en plus de spécialiser certains attributs sur les *noeuds* (attributs spécifiques à SMIL), fournit aussi un ensemble de méthodes plus spécialisées pour le cadre des documents multimédias (méthodes *map* ou *play* par exemple). La fonction *map* permet d'afficher le document à un instant précis de la présentation sans déclencher leur exécution.

Cependant cette interface répond aux besoins des systèmes de présentation de documents multimédias, mais est peu adaptée à un processus d'édition. Cette interface ne contient pas par exemple, les méthodes *addOpérateur*, *changeOpérateur* utiles lors de l'édition de la structure temporelle d'un document SMIL.

Structure de document dans Kaomi :

La structure de document dans Kaomi respecte l'interface DOM (structure et méthode) et se compose essentiellement de quatre parties :

- Le noeud document : cet objet sert à stocker toutes les informations propres à un document multimédia. Il contient en particulier les structures temporelle et spatiale, ainsi que la liste des objets multimédias du document.
- La structure de données temporelle : elle permet de sauvegarder les informations temporelles du document.
- La structure de données spatiale : elle permet de sauvegarder les informations spatiales du document.
- Une liste d'élément: un élément est soit un noeud document, soit un élément multimédia.

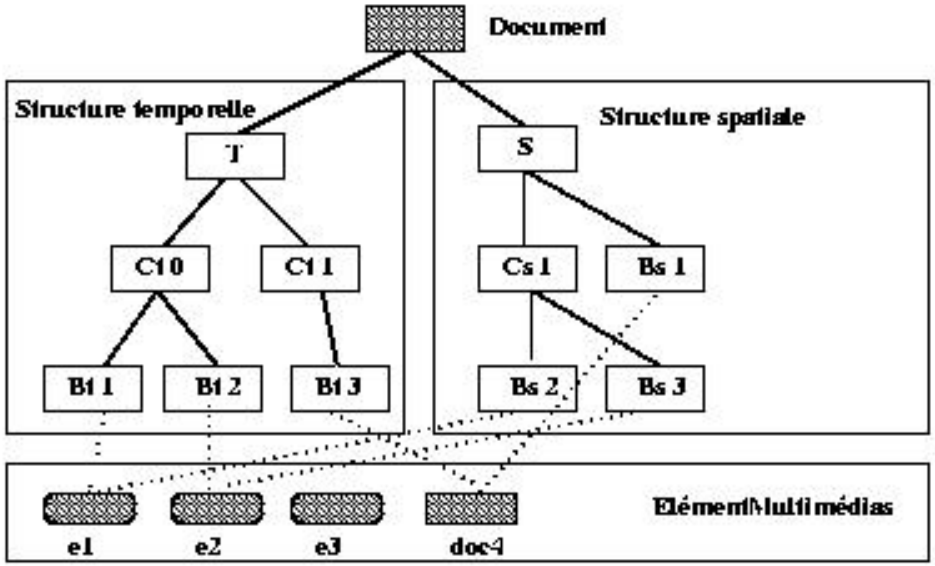


Figure IV-11 : Structure d'un document dans Kaomi

Les structures temporelle et spatiale sont constituées d'objets composites (Ct, Cs) et d'objets basiques (Bt, Bs). Les objets composites permettent de

regrouper un ensemble d'objets, les objets basiques sont les feuilles de la structure et sont liés aux objets multimédias. Nous allons présenter de manière plus précise la structure temporelle dans la section 7.

La dimension spatiale ne sera pas décrite. En effet, même si les paradigmes d'édition offerts finalement à l'auteur diffèrent (édition directe dans la vue de présentation), les structures de données et les mécanismes mis en oeuvre sont très proches de ceux utilisés pour la dimension temporelle.

7 Le modèle temporel de Kaomi

L'objectif du modèle temporel de Kaomi est de permettre de représenter un large éventail de formalismes de documents multimédias (absolu, relationnel, événementiel). Ce modèle temporel repose essentiellement sur des éléments basiques (Bt) composés de 3 attributs (Début, Fin, Durée) et d'éléments composites composés de cinq types d'informations attachés à chaque composite temporel Ct (Figure IV-12) :

- L'opérateur temporel placé par l'auteur ou par défaut en fonction du langage. Par exemple, l'opérateur *inside* sera associé à tous les composites temporels du langage Madeus.
- Les attributs temporels qui permettent de modéliser les instants de début, fin de l'objet, ainsi que la durée de l'objet (section 7.1).
- Les tables de relations qui permettent stocker les informations de composition que l'auteur place entre les objets multimédias (section 7.2).
- Les événements qui permettent de stocker les informations de type événementiel entre les objets, ces événements seront traduits en rélems (section 7.2.3).
- Les opérateurs, relations et événements seront traduits en termes de relations élémentaires. Nous appellerons par la suite les relations élémentaires des *rélems*. Le processus de traduction sera présenté dans la section 7.2.2.
- Le graphe temporel qui est une représentation des informations temporelles des objets qui forment le composite. Le graphe temporel sera construit automatiquement à partir des rélems. Les graphes sont utiles pour la présentation du document dans la vue d'exécution [Sabry99], et pour l'affichage d'informations dans la vue temporelle [Tardif97] (section 7.3). Ils sont aussi utiles pour le formatage et la vérification de cohérence [Layaïda97].

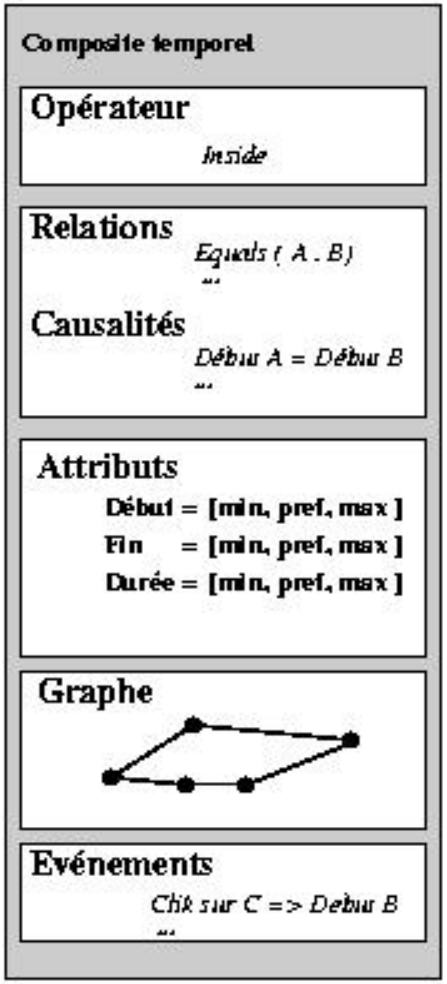


Figure IV-12 : Informations temporelles associées à un composite

Nous maintenons une redondance entre les informations. Nous gardons les relations dans la structure du fait que lors du processus d'édition nous avons besoin de maintenir le lien entre la relation spécifiée par l'auteur et les rélems. En effet, lorsque l'auteur supprimera une relation, nous aurons besoin de supprimer les rélems générés par cette relation.

Nous allons maintenant détailler ces informations temporelles et voir comment les relations temporelles mises entre les objets modifient les attributs temporels du document et permettent la construction du graphe.

7.1 Les propriétés temporelles

Les attributs temporels permettent de modéliser les propriétés temporelles d'un élément multimédia. Ces attributs sont *début*, *fin* et *durée*. Ces attributs ne sont pas définis par une valeur mais par un intervalle de valeurs et une valeur *préférée*. Chaque attribut est donc défini par trois paramètres : [borne inférieure, valeur préférée, borne supérieure].

Chaque paramètre peut avoir une valeur *initiale* et une valeur *formatée*. La valeur initiale est celle qui est donnée par l'auteur ou celle qui a été mise par défaut par le système auteur. La valeur formatée est celle qui a été calculée par le système en fonction du contexte de l'élément temporel, c'est-à-dire en fonction des relations qui sont dans le document et qui peuvent modifier sa durée. Par exemple, si on a deux objets A et B, avec initialement pour intervalle de durées [1,3,4] et [2,5,9], et si l'auteur met une relation d'égalité entre les deux objets alors les valeurs formatées des deux objets seront dans ce cas : [2,3,4] et [2,3,4] car le système enlève automatiquement les valeurs qui n'appartiennent à aucune solution. Les techniques utilisées seront présentées dans le chapitre V.

7.2 Opérateurs, relations et événements dans Kaomi

Dans le modèle temporel de Kaomi, on peut associer à la fois une sémantique d'arbre d'opérateurs, de relations et d'événements à la structure temporelle.

Un opérateur est une relation qui lie tous les fils d'un noeud composite. Par exemple, si on associe l'opérateur *pendant* à un composite temporel, tous les fils devront se jouer pendant le père, le père sera une boîte englobante temporelle de ses fils. D'autres opérateurs fréquemment utilisés sont les opérateurs *séquence* et *parallèle* (cf. SMIL). Les opérateurs seront traduits en relations entre les éléments multimédias.

Les relations temporelles sont placées entre les fils d'un noeud composite, elles peuvent être unaires ou binaires. Les relations permettent de définir le comportement temporel des objets. Ces relations peuvent compléter l'opérateur associé au composite temporel. Les relations temporelles sont stockées sur les noeuds composites. Dans Kaomi, le modèle de relation est extensible et modifiable, c'est-à-dire qu'il n'y a pas un ensemble prédéfini de relations. Les relations sont définies de manière abstraite, et sont complétées par des informations sur leur sémantique exprimées en rélems. C'est ce mécanisme que nous présenterons dans la section (7.2.2).

Les événements dont les actions modifient le comportement ou la présentation d'éléments frères de l'objet ayant déclenché l'événement seront eux aussi traduits en rélems. Ce mécanisme sera présenté dans la section 7.2.3

Le mécanisme général de traduction des opérateurs, des relations et des événements sera illustré dans la section 7.2.4.

7.2.1 Les relations élémentaires de Kaomi

Les rélems sont les entités de base du modèle temporel de Kaomi. Ils représentent les informations temporelles élémentaires. C'est une extension des relations d'instants déduites des relations d'intervalles d'Allen car elles contiennent en plus des relations sur la durée des objets.

Un rélem est essentiellement composé de trois informations : un ou deux attributs temporels, une opération et un paramètre. Kaomi permet de définir

deux types de rélems : les rélems unaires et les rélems binaires selon qu'ils portent sur un ou deux attributs temporels.

Rélem unaire, il est défini par trois attributs **:**

- Attribut temporel : un attribut temporel qui est soit le début d'un objet temporel, soit sa fin.
- Opération : dans le cas d'un rélem unaire l'opération peut être un événement (Event) ou un décalage dans le temps d'un instant de début ou de fin (Δ).
- Paramètre qui correspond par exemple au délai dont on décale un instant dans le temps.

Dans l'exemple de la Figure IV-13, on peut voir trois rélems unaires. Le premier a pour effet d'avancer la fin de l'objet temporel A de 12 secondes, la deuxième retarde le début de l'objet temporel B de 5 secondes. Le troisième exemple est un événement qui sera déclenché à la fin de l'objet A. Les événements seront décrits de manière plus précise dans la section 7.2.3.

Instant temporel	Opération	Paramètre
Fin (Objet Temporel A)	Δ	-12s
Début (Objet Temporel B)	Δ	5s
Fin(Objet Temporel A)	Event	événement

Figure IV-13 : Exemples de rélems unaires

Rélem binaire, il est défini par trois attributs **:**

- Deux attributs temporels des objets impliqués dans le rélem.
- Opération : une relation soit entre deux instants temporels (t_1, t_2) soit entre deux durées (dur_1, dur_2). Cette relation peut être :
  - $t_1 > t_2$: l'instant t_1 doit être après l'instant t_2.
  - $d_1 > d_2$: la durée d_2 doit être plus courte que la durée d_1.
  - $t_1 < t_2$: l'instant t_1 doit être avant l'instant t_2.
  - $d_1 < d_2$: la durée d_1 doit être plus courte que la durée d_2.
  - = : les deux instants ou les deux durées doivent être égaux.
  - $\Rightarrow$: l'instant t_1 provoquera l'instant t_2.
  - $\Leftarrow$: l'instant t_2 provoquera l'instant t_1.
- Un paramètre qui correspond au paramètre éventuel de la relation. Par exemple l'instant 1 doit être *t* secondes avant l'instant 2.

Dans la Figure IV-14, on peut voir trois exemples de rélems binaires. Le premier indique que les deux objets temporels A et B ont la même durée. Le

deuxième indique que la *Fin* de l'objet temporel A est 12s après le *Début* de l'objet temporel B. Enfin, le troisième exemple indique que lorsque l'objet temporel A se terminera, alors le système arrêtera l'objet temporel B.

Attribut temporel	Attribut temporel	Opération	Paramètre
Durée(Objet Temporel A)	Durée (Objet Temporel B	=	
Fin (Objet Temporel A)	Début (Objet Temporel B)	>	12
Fin (Objet Temporel A)	Fin (Objet Temporel B)	⇒	

Figure IV-14 : Exemples de rélems binaires

7.2.2 Spécification des opérateurs et des relations dans Kaomi

Le concepteur de l'environnement auteur définit une liste de relations qui lui semblent nécessaire pour l'auteur en définissant pour chaque relation la liste des relations élémentaires (rélems) qui décrit la sémantique de cette relation. Cependant, si l'auteur lui même souhaite agrandir cette liste il peut le faire de la même manière que le concepteur de l'environnement.

Pour cela, nous avons utilisé le système des ressources fourni par le langage Java (voir section 5.1.3) ainsi Kaomi fournit un fichier de ressources qui décrit la sémantique des relations et qui permet de mettre à jour les informations dans les structures de données de Kaomi.

Dans la Figure IV-15, on peut voir un exemple de ressource qui décrit la relation *equals*. La sémantique des relations s'exprime en terme de rélems.

```
#Sémantique de la relation equals

#nombre de rélems engendrés par la relation
equals.nbRélem = 3

#nombre de paramètres de la relation
equals.param=2

# Dur(A) = dur (B)
equals.1.attribut1 = DURATION
equals.1.attribut2 = DURATION
equals.1.RélemRelation=EQUALS



# Begin(A) = Begin(B)
equals.2.attribut1 = BEGIN
equals.2.attribut2 = BEGIN
equals.2.RélemRelation=EQUALS

# End(A) = End(B)
equals.3.attribut1 = END
equals.3.attribut2 = END
equals.3.RélemRelation=EQUALS
```

Figure IV-15 : Ressource décrivant une relation

La description des opérateurs repose sur le même principe de ressource. Le concepteur de l'environnement auteur décrit la sémantique des opérateurs en terme de relations. L'utilisation des relations permet de rendre la description des opérateurs plus simple. La sémantique d'un opérateur peut être définie selon le type de l'élément inséré.

Dans la Figure IV-16, on peut voir un exemple de ressource décrivant l'opérateur *Inside*.

```
# Sémantique de l'opérateur inside

# Sémantique de l'opérateur pour un élément multimédia de type délai
inside.delay.number=1

#L'objet n'a pas de relation avec son père s'il est de type délai
inside.1.delay.relation=null

# Sémantique de l'opérateur pour un élément multimédia de type média
inside.media.number=1

#L'objet a une relation inside avec son père
inside.1.media.relation=inside_by
inside.1.media.first_object=father
inside.1.media.second_object=current
```

Figure IV-16 : Sémantique d'un opérateur

7.2.3 Les événements de Kaomi

Généralités :

Nous avons vu dans le chapitre II (section 3.2) que de nombreux modèles de documents utilisent des événements. Nous avons choisi de garder dans la structure interne de Kaomi la même approche que ces modèles et les événements s'expriment sous une forme proche de règles : condition /action.

Dans le modèle temporel de Kaomi, les événements sont stockés sur les noeuds composites de la structure temporelle. De manière à favoriser la localité des informations, les événements sont stockés sur le premier élément temporel qui contient dans sa descendance tous les objets intervenant dans la spécification de l'événement. Dans le cas extrême, cela peut être la racine du document.

Dans la mesure du possible, les événement sont automatiquement traduits en rélems.

Définition :

Un événement de façon générale peut être défini par six attributs :

1. *Source de l'événement* : permet de connaître l'origine de l'émission de l'événement. Cet attribut peut prendre les valeurs :

- utilisateur, pour une interaction avec le document,
- application qui donne des informations au document (configuration de la machine, changement d'état de certains paramètres,)

- objet du document, par exemple pour notifier sa fin.

1. *Destination de l'événement* : cet attribut peut prendre trois valeurs:

- Application, pour mettre à jour des données générales liées à l'utilisateur par exemple.
- Utilisateur, par exemple pour afficher des messages d'erreurs ou des messages informatifs.
- Un objet du document, c'est par exemple le cas pour interrompre un objet.

1. *Nombre d'occurrences* : cet attribut peut prendre deux types de valeur : un *entier* qui indique le nombre de fois où l'événement se produit dans le cas où celui-ci serait prédictible, ou alors *infini* dans le cas contraire. Par exemple, lorsque l'utilisateur clique sur un bouton *pause/resume* l'événement associé peut être déclenché un nombre infini de fois ; par contre, dans le cas du bouton *quitter l'application*, celui-ci ne peut intervenir qu'une fois.
2. *Date des occurrences* : une liste de dates correspondant aux différentes occurrences quand celles-ci sont prédictibles, une liste de variables dans le cas contraire.
3. *Une condition***:** une expression booléenne qui permet de tester un ensemble d'attributs sur des objets du document, ou des paramètres de l'application.
4. *Une liste d'actions* : la liste des actions à effectuer sur la destination de l'événement lorsque la source émet l'événement et que la condition est évaluée à vrai.

Exemple de traduction :

Pour montrer que cette structure de données est suffisamment générale on va s'intéresser à la traduction des événements MHML dans notre formalisme.

Par exemple :

Evénement MMHL :

Link L_1 indique à quel groupe est associé l'événement.
Type_Event = E_1
Nom_Condition = {C_1,C_2, C_3) où C_i est une expression booléenne.
Action = {A_1, A_2, A_3}

Dans notre formalisme :

Source = E_1
Destination = L_1
NbOccurrences = infini
Dates d'occurrences = {t_1, t_2, , t_n} avec n infini, et t_i une variable.
Conditions = C_1 & C_2 & C_3
Actions = {A_1, A_2, A_3}

Traduction en rélems :

La traduction des événements en rélems se décompose en deux cas :

- L'événement est traduisible dans un rélem non événementiel. Par exemple, un événement de la forme :

événement E1 = { Source = $Média_1$
Destination = $Média_2$
NbOccurrences = 1
Dates d'occurrences = {14}
Conditions = Fin($Média_2$)
Actions = Démarrer($Média_1$) }

sera traduit par le rélem suivant : Fin(A) ⇒ Début(B). La traduction de l'événement dans ces rélems permettra de profiter des services de manipulation dans la vue temporelle et d'appliquer des mécanismes de formatage sur les objets A et B.

- L'événement n'est pas traduisible dans un rélem non événementiel. Dans ce cas, il sera traduit à l'aide du rélem Event (avec *E1* en paramètre), l'événement sera alors traité de manière spécifique lors de la présentation, et l'environnement d'édition n'offrira pas de service de manipulation dans la vue temporelle.

Notons par exemple que dans le cas du langage MHML où tous les comportement s'expriment sous forme événementielle, on arrive à isoler un certain nombre de comportements non événementiels. Nous verrons lors de la présentation de MHML-Editeur (Chapitre VI) ces différents cas ainsi qu'une expérience menée pour généraliser ces travaux au cadre des événements imprédictifs.

7.2.4 Traduction des relations en relations élémentaires

Nous allons maintenant nous intéresser au processus général de traduction des relations en rélems lors de la création d'un document par un auteur.

Kaomi utilise le nom de la relation mise par l'auteur pour construire automatiquement les rélems à partir des fichiers de ressources. L'exemple de la Figure IV-17 illustre ce processus.

Dans un premier temps, l'auteur va créer deux objets A et B. Lors de cette création, l'application initialise les attributs temporels des deux objets. Ensuite l'auteur insert les deux objets dans un objet composite scène défini préalablement. Nous avons associé l'opérateur *inside* à cet objet scène. Les deux objets sont donc en relation *inside* avec leur père. Le système utilise la ressource qui décrit l'opérateur *inside* pour mettre la relation adaptée, et ensuite la ressource qui décrit la relation pour mettre les rélems correspondants. Dans l'exemple, l'utilisateur met ensuite une relation e*quals* entre les deux objets, relation qui sera traduite en rélems par le même mécanisme.

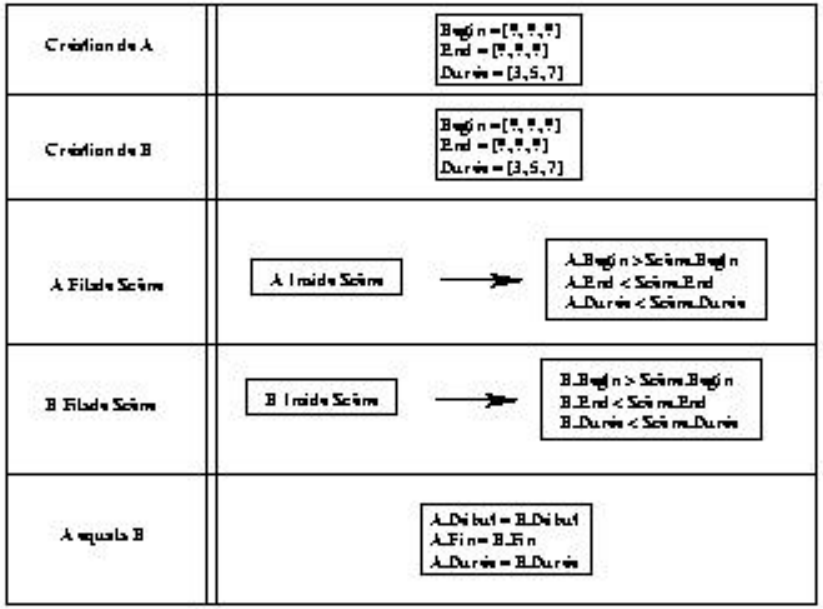


Figure IV-17 : Mécanisme de mise à jour des rélems

7.3 Construction des graphes temporels

Une des utilisations des rélems est la construction des graphes.

Un graphe est composé d'arcs qui représentent les objets (leur durée) et de noeuds qui représentent les instants (de début ou de fin) des objets.

Nous allons dans un premier temps présenter le mécanisme de base pour la création des graphes temporels (section 7.3.1) avant de présenter les trois difficultés rencontrées lors de cette création :

- traduction des rélems pour les relations de causalités (section 7.3.2) ;
- retrait de rélems (section 7.3.3) ;
- construction de graphes hiérarchiques (section 7.3.4).

7.3.1 Mécanisme de base

L'édition d'un document se fait par ajout/retrait d'objets et de relations, d'opérateurs ou d'événements. Ces différentes informations, comme nous l'avons introduit précédemment seront traduites en rélems. Par conséquent la construction du graphe se fait lors de l'ajout ou du retrait d'une information temporelle. Le graphe est construit automatiquement lors de la construction des noeuds temporels et de l'insertion de relations.

À chaque création d'objet composite, on crée un nouveau graphe. À chaque création d'objet temporel élémentaire ou de composite, on crée un arc

dans le graphe de son père. Le noeud de début de l'arc représente l'instant de début de l'objet, le noeud de fin représente l'instant de fin de l'objet temporel.

À chaque rélem créé, on réalise une opération dans le graphe (Figure IV-18). Par exemple, lors de l'insertion d'un rélem qui entraîne l'égalité de deux instants, la fusion des deux noeuds temporels représentant ces deux instants est effectuée.

Rélem	Opérations
$t_1 = t_2$	Fusion de noeuds
$t_1 < t_2$	Insertion d'un délai entre deux noeuds
$t_1 > t_2$	Insertion d'un délai entre deux noeuds

Figure IV-18 : Liens rélem / opération dans le graphe

Les rélems Event, et ceux qui définissent des relations sur les durées ne sont pas traduisibles en termes de graphe. Les services reposant sur le graphe (gestionnaire de présentation par exemple) pourront accéder à ces informations via les tables de rélems.

Dans la Figure IV-19, on peut voir la construction de ce graphe lors de l'insertion de deux relations différentes. Tout d'abord, lors de la création d'un objet, le système construit un arc et deux noeuds qui représentent l'objet nouvellement créé. L'auteur insère ensuite une relation *equals*, qui a pour effet de fusionner les noeuds représentant les instants de début et de fin des objets. L'auteur insère finalement une relation *before*, dans ce cas le système insère un délai entre les deux objets.

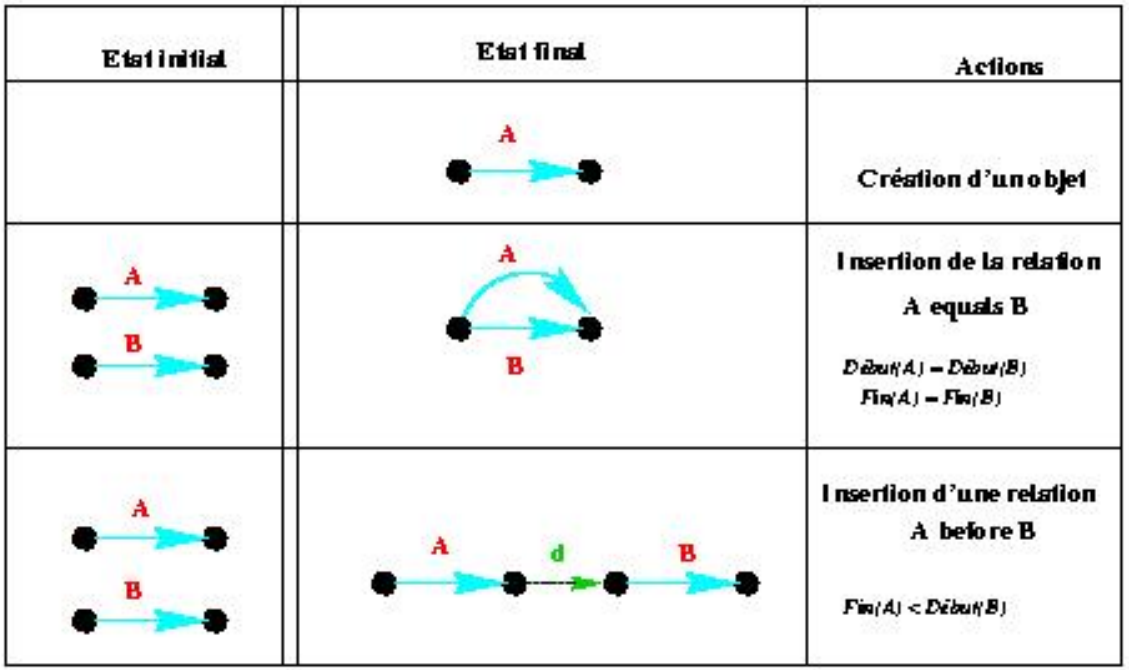


Figure IV-19 : Construction du graphe temporel

L'exemple de la Figure IV-20 est plus complet. Dans la partie gauche, on visualise la liste des relations, et dans la partie droite on visualise le graphe résultant de cette spécification.

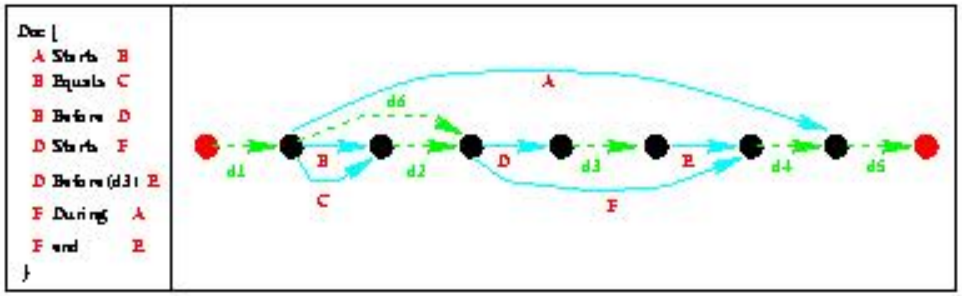


Figure IV-20 : Graphe temporel résultant d'une spécification

7.3.2 Construction du graphe avec les rélems de causalité

L'insertion des rélems de causalité a posé quelques problèmes lors de la construction du graphe.

En effet, prenons l'exemple du rélem Fin(A)⇒ Fin(B) signifiant que la fin de l'objet A entraîne la fin de l'objet B indépendamment de la valeur prévue initialement pour B.

La première solution qui pourrait être choisie pour représenter ces deux objets dans le graphe serait de fusionner les noeuds de fin, comme pour le rélem "=". Cette représentation simple pourrait être satisfaisante pour la machine de présentation. Elle poserait néanmoins un problème aux formateurs se basant sur le graphe. En effet, statiquement ces deux objets n'ont pas la même durée, ils ne peuvent donc pas être représentés par des arcs ayant les mêmes instants de début et de fin. La solution que nous avons choisie et de définir deux arcs pour les objets étant interrompus dynamiquement. Un arc représente la valeur effective prise lors de la présentation, et un arc représente la valeur prévue statiquement. De manière à assurer la cohérence du graphe, nous introduisons un délai entre les fins prévues et les fins réelles des objets interrompus.

Rélem	Opérations
$t_1 \Rightarrow t_2$	Insertion de deux arcs fictifs : ● un arc qui représente la durée prévue de l'objet auquel appartient t_2 ● un délai qui relie la fin (ou le début) prévue (sans interruption) avec la fin (ou le début) réelle de l'objet L'arc initial représente maintenant la durée réelle de l'objet.
$t_1 \Leftarrow t_2$	Insertion de deux arcs fictifs : ● un arc qui représente la durée prévue de l'objet auquel appartient t_1 ● un délai qui relie la fin (ou le début) prévue (sans interruption) avec la fin (ou le début) réelle de l'objet L'arc initial représente maintenant la durée réelle de l'objet.

Figure IV-21 : Liens rélem / opération dans le graphe

Dans le cas particulier où deux instants ont les deux relations $\Rightarrow$ et $\Leftarrow$, alors le système doit assurer qu'un des deux délais introduits ait une durée nulle. Ceci afin de conserver la sémantique de la relation.

Dans la Figure IV-22 nous pouvons voir un exemple de construction de graphe avec des relations causales. L'auteur initialement crée trois objets A, B et C. Il insert ensuite une relation *parmin* entre les objets A et B. Dans la figure, nous avons détaillé la construction du graphe pour les quatre causalités engendrées par la relation *parmin*. L'auteur insert ensuite une relation *meet* entre les objets A et C. Notons que cette nouvelle relation sera

insérée à la suite de l'arc représentant la durée réelle de A.

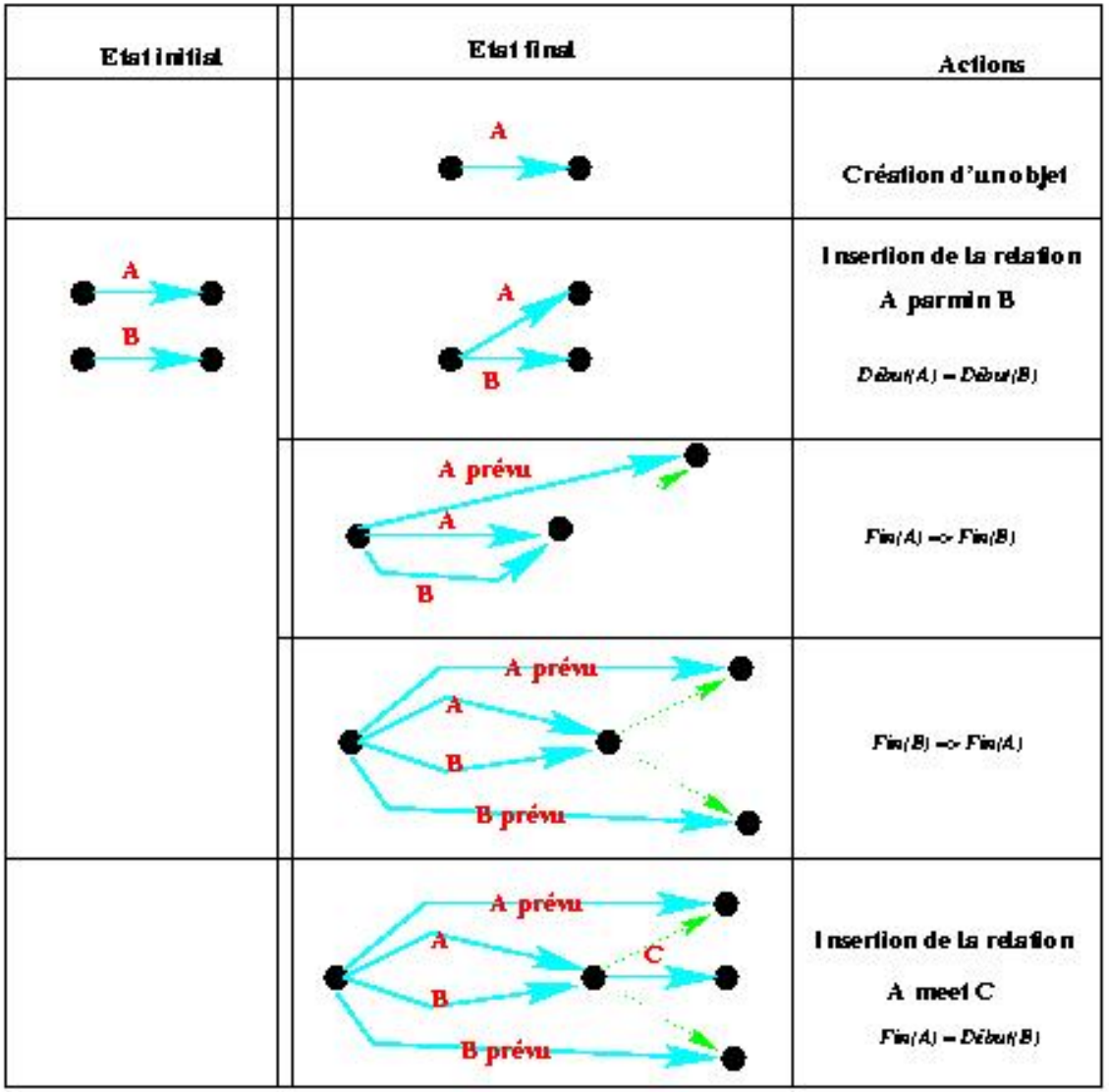


Figure IV-22 : Construction de graphe avec les relations causales

7.3.3 Retrait de rélem

Lors du retrait d'une relation, d'un événement ou de la modification d'un opérateur, le système doit supprimer des rélems. Cela se fait facilement du

fait que nous gardons la liste des rélems créés par chaque relation, opérateur ou événement.

Lors du retrait d'un rélem le graphe doit être mis à jour. Cette opération est plus complexe, car lors du retrait d'un rélem, il faut être capable de défusionner les noeuds et d'étendre les intervalles de valeurs des durées des objets.

Le choix que nous avons fait est de réinitialiser tous les arcs du composite avec leurs durées initiales.

On peut noter de plus que l'introduction des relations causales, comme le parmin, pose quelques problèmes à l'édition. Prenons le scénario suivant :

- Création d'un objet composite C.
- Création d'un objet A de durée comprise entre 6 et 8.
- Création d'un objet B de durée 5
- Insérons A et B comme fils de C.
- Insérons une relation *parmin* entre les objets B et A. Cela a pour effet d'affecter 5 comme durée réelle sur l'objet A.
- Spécifions une durée de 5 sur C. Cette spécification est cohérente car la durée des objets sous C est de 5.
- Maintenant enlevons la relation *parmin*. Les objets A et B reprennent leurs valeurs initiales. Le document est incohérent du fait qu'un des fils de C a une longueur supérieure à 5.

Il faut donc faire attention lors du retrait des relations causales car le document peut être incohérent. Il est à noter que le retrait d'une relation non causale ne peut jamais mettre le document dans un état incohérent. C'est une des propriétés intéressantes des CSP (voir chapitre V section 3.3).

7.3.4 Construction de graphes hiérarchiques

La dernière difficulté dans la construction des graphes est liée à l'introduction des graphes hiérarchiques pour modéliser la présence de composite dans la structure temporelle.

L'introduction de la hiérarchie a été réalisée de la manière suivante :

- Un graphe est créé pour chaque composite et il représente l'enchaînement temporel des éléments du composite.
- Chaque élément (basique ou composite) est représenté par un arc dans le graphe de son père.
- Les éléments composites sont aussi représentés par un arc dans leur propre graphe de manière à assurer une cohérence entre les informations contenues dans les différents graphes. Un mécanisme spécifique assurera la cohérence entre les deux arcs représentant les éléments composites. De plus, lors de l'insertion de chaque élément, le système créera deux *arcs d'insertion* pour relier l'arc représentant l'élément au début et à la fin de l'arc représentant le composite contenant le graphe.

Les graphes ainsi obtenus ont la propriété de n'avoir qu'un noeud de début et qu'un noeud de fin. De plus, tous les arcs sont reliés à l'arc représentant l'élément composite.

La création des *arcs d'insertion* nécessite, lors du formatage et de l'analyse, l'existence d'un mécanisme qui vérifie qu'au moins un des arcs d'insertion sortant du premier noeud du graphe et qu'un des arcs d'insertion entrant du dernier noeud du graphe aient une durée de 0. Ceci afin de conserver le fait

que chaque noeud du graphe peut être assimilé à un instant temporel. Et donc, que les instants de début et de fin d'un graphe correspondent aux instants de début du premier objet temporel et de fin du dernier objet temporel du graphe.

Dans la Figure IV-23 nous pouvons voir un exemple de graphes hiérarchiques. Le composite C qui est un élément composite est donc représenté dans deux graphes.

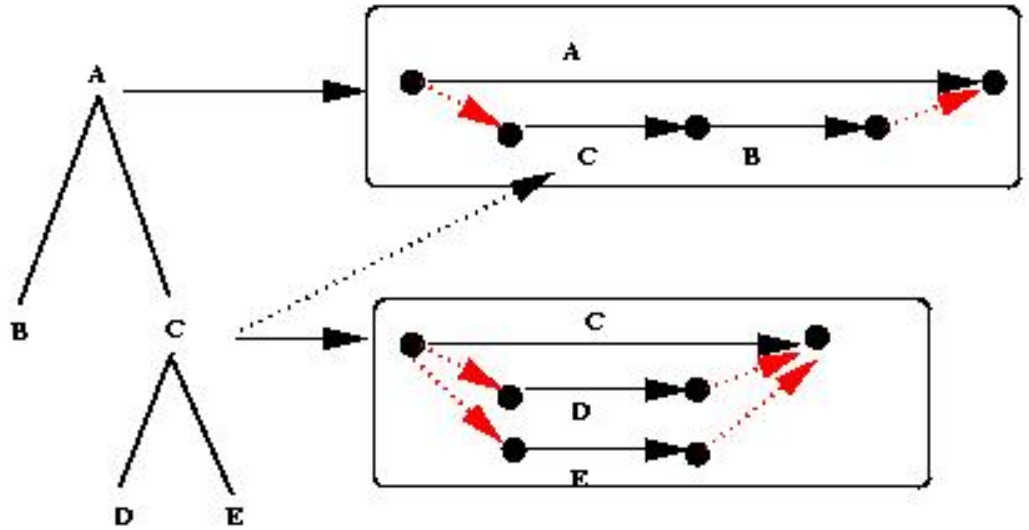


Figure IV-23 : Construction de graphes hiérarchiques

7.4 Bilan du modèle de document de Kaomi

Le modèle de document proposé dans Kaomi couvre les besoins énoncés dans le chapitre III pour réaliser un environnement idéal. En effet, le modèle d'édition proposé permet principalement de :

- Structurer le document ;
- Spécifier un document adaptable ;
- Définir des événements entre les objets ;
- Définir des relations entre les objets.

Ce modèle cherche aussi à couvrir une famille large de langages multimédias source. Dans ce but, cette structure se base essentiellement sur la notion de relations élémentaires. Nous validerons ce modèle de document dans l'implémentation d'environnement auteur pour les langages SMIL, Madeus et MHML.

8 Fonctions et services d'édition fournis par Kaomi

Kaomi est une boîte à outils qui vise à construire des environnements auteur. Ces environnements bien qu'ayant des formalismes différents peuvent fournir un ensemble de services communs ou reposant sur les mêmes opérations de base. Kaomi va, dans ce but, fournir un ensemble d'opérations d'édition le plus générique et le plus extensible possible. Nous allons présenter ce principe dans cette section.

Une opération d'édition est une opération de l'auteur qui entraîne la modification d'un ou plusieurs attributs sur un objet du document ou encore une modification de sa structure.

Ainsi, toute opération d'édition peut se ramener à une ou plusieurs opérations élémentaires qui peuvent être :

- la création / suppression d'un objet (section 8.1);
- la modification d'un attribut (section 8.2);
- l'ajout / retrait d'une relation /un événement / un opérateur (section 8.3);
- la création ou la suppression de groupes (section 8.4).

Ces opérations d'édition que nous allons définir reposent sur l'édition du format pivot que nous venons de présenter. Ce sont donc des opérations d'édition qui s'effectuent pour l'auteur au travers du formalisme d'édition de l'environnement auteur qui est une extension de celui du langage de sauvegarde.

Chaque opération d'édition impliquera une vérification de cohérence du document. De ce fait, les mécanismes vérifiant la cohérence devront être très performants. Les techniques mises en oeuvre pour cela seront présentées dans le chapitre V.

8.1 La création d'objets

Dans Kaomi, le concepteur d'application a à sa disposition un ensemble de méthodes qui permettent de créer des médias ou des groupes. Ces méthodes se basent sur l'objet sélectionné pour insérer le nouvel objet dans la hiérarchie. Si l'objet sélectionné est un groupe, l'objet est inséré dans le groupe, sinon l'objet est inséré dans le groupe contenant l'objet sélectionné. Dans le cas où il n'y a pas d'objet sélectionné, l'objet est inséré à la racine du document.

8.2 L'édition d'attributs dans Kaomi

Les opérations de modification d'un attribut dans Kaomi peuvent se faire de deux manières :

- Modification de l'attribut via une palette, dans ce cas, le système recalcule les informations nécessaires pour rendre le document cohérent, et calcule donc une nouvelle solution. La palette d'attributs est configurable en fonction des souhaits du programmeur (voir section 9.6).
- Modification de l'attribut par manipulation directe dans la vue de présentation ou la vue temporelle. Dans ce cas, le système calcule un ensemble de solutions intermédiaires tout au long de la manipulation de l'auteur.

Dans l'exemple de la Figure IV-24, on peut voir un exemple d'opération d'édition. L'auteur par une opération via la palette d'attributs a modifié un attribut temporel. Lors de cette modification le système doit assurer la cohérence du système et propager les nouvelles informations aux différentes vues.

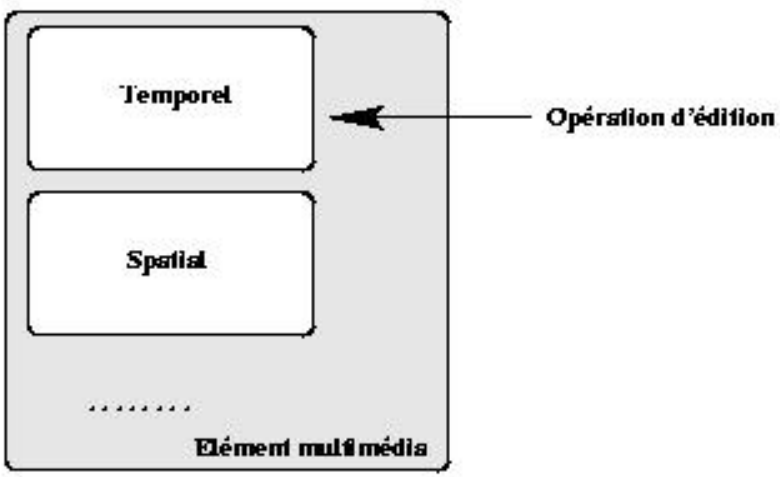


Figure IV-24 : Edition d'un attribut temporel dans Kaomi

Certaines propriétés de cohérence sont indépendantes du langage comme la cohérence qualitative. Les mécanismes de vérification de celles-ci sont offerts de manière générique par Kaomi (voir Chapitre V). Cependant d'autres propriétés sont dépendantes des langages comme dans le langage SMIL où deux objets présents temporellement au même instant ne peuvent partager la même région spatiale. De telles vérifications doivent être prises en charge par le concepteur de l'environnement auteur.

8.3 Edition de relations

Le deuxième type d'opération d'édition que peut effectuer l'auteur est l'édition de relations. Chaque langage a son ensemble de relations avec sa sémantique qui lui est propre. L'outil auteur propose donc une palette de relations, relations qui sont à la fois dépendantes du langage et du système auteur (chapitre III section 2.2.2).

Dans le cas du langage Madeus-97, l'auteur aura à sa disposition dans la palette temporelle les différentes relations du langage ainsi que par exemple les 4 relations supplémentaires présentées dans l'environnement idéal pour faciliter l'édition.

Le mécanisme de construction de cette palette est similaire à celui de la palette d'attributs, c'est-à-dire qu'il est basé sur un mécanisme de ressources. L'auteur peut ainsi sélectionner des objets dans une vue et insérer une relation entre eux.

L'édition des événements et des opérateurs fonctionne sur le même principe.

8.4 La création de groupe

Kaomi fournit un ensemble de fonctions permettant de créer et détruire des groupes. Ces fonctions permettent notamment à l'auteur d'éditer la structure temporelle et spatiale de son document.

La création de groupe peut se réaliser simplement. L'auteur sélectionne un objet composite et clique sur un menu ou une icône permettant de créer un nouveau groupe. L'environnement auteur insert alors un nouveau groupe à l'intérieur de l'élément composite sélectionné. L'auteur pourra ensuite insérer de nouveaux éléments dans ce composite. Aussi simplement la destruction d'un groupe ne contenant plus d'objets se fait par sélection de l'élément à supprimer puis par appel de la fonction détruire par l'intermédiaire d'un menu ou d'une icône.

Cependant, en cours d'édition l'auteur attend des fonctions de plus haut niveau. Par exemple, il peut avoir envie de sélectionner un ensemble d'objets pour les grouper. Se pose alors pour le concepteur de l'outil auteur la question de la sémantique de cette opération. En effet, cette opération soulève de nombreuses questions :

- Que faire des relations existantes entre les éléments sélectionnés et non sélectionnés ?
- Que faire de la sémantique des groupes (opérateurs) dans lesquels étaient les éléments sélectionnés ?

De nombreuses solutions sont possibles. Nous avons fait un choix de comportement qui est, bien entendu, restrictif. Il serait souhaitable de pouvoir définir la politique prise lors de cette opération pour chaque environnement auteur par un mécanisme de ressource.

Le choix pris par défaut est le suivant :

- Le nouveau groupe est inséré au niveau du premier ancêtre commun des objets sélectionnés.
- Nous conservons l'opérateur attaché au premier objet sélectionné.
- Nous supprimons les relations liant les objets sélectionnés et les objets non sélectionnés.
- Nous conservons les relations reliant les objets sélectionnés.

Dans l'exemple de la Figure IV-25, l'auteur a sélectionné un ensemble de quatre objets (C,E,G,H). Il a ensuite demandé de grouper ces quatre objets. L'environnement auteur réalise donc cette opération. Cela se traduit par : la destruction des relations (Rel(A, B), Rel(D, E), Rel(F, G)), la création d'un nouvel élément composite (Nouveau composite), la conservation de l'opérateur du premier objet sélectionné (*Inside*) et la conservation de la relation Rel(G, H) qui liait deux objets sélectionnés.

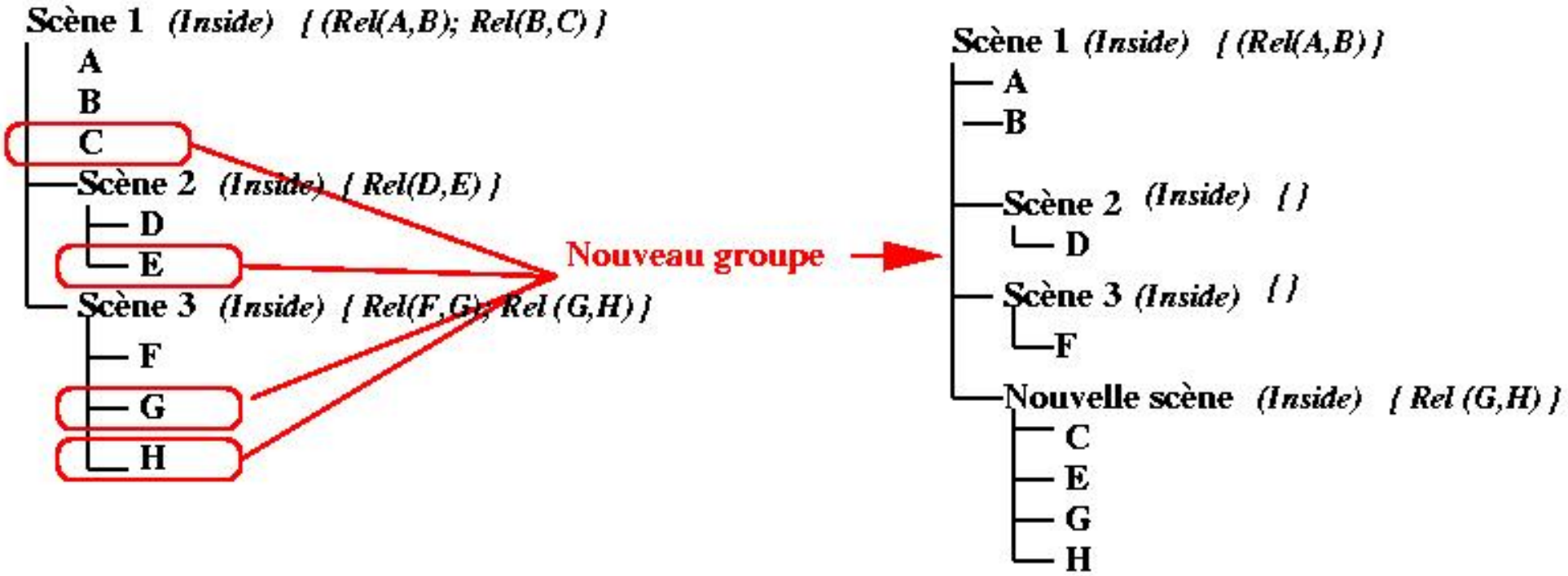


Figure IV-25 : Exemple de création de groupe

9 Vues et synchronisation

Comme nous avons pu le dire précédemment, Kaomi fournit un ensemble de vues, conformes à celles qui sont spécifiées dans la section 2.3. Nous allons, dans un premier temps, présenter le mécanisme d'édition avec une vue (section 9.1) ainsi que le principe général de construction d'une vue (section 9.2) avant de présenter les principales vues de Kaomi :

- Vue de présentation (section 9.3) ;
- Vue hiérarchique (section 9.4) ;
- Vue résumé (section 9.5) ;
- Vue attributs (section 9.6) ;
- Vue temporelle (section 9.7).

Les différentes fonctions des vues seront illustrées avec leur mise en oeuvre dans les environnements auteur réalisés avec Kaomi (SMIL-Editeur, Madeus-Editeur,) dans le chapitre VI.

9.1 Mise à jour des vues

Nous avons présenté dans la section 5.2 le processus général de construction d'une vue ainsi que la synchronisation entre les différentes vues avec le document de référence. Ces mécanismes reposent essentiellement sur trois points :

- Construction des documents étendue par transformations du document de référence (ce mécanisme sera illustré dans chacune des vues présentées).
- La synchronisation des attributs qui assure que lorsqu'un attribut est modifié, alors toutes les vues sont informées et modifiées.
- La synchronisation des structures qui permet à chacune des vues d'être informée d'une modification de la structure (ajout/retrait d'objet par exemple).

Ces mécanismes sont implémentés à la fois dans les différentes classes de base qui définissent les attributs, mais aussi dans la classe *Document*. Les classes *referenceDocument* et *extendedDocument* héritent de cette classe *Document*, et profitent de ce fait de ces mécanismes.

Dans la Figure IV-26, nous étendons le schéma présenté dans la Figure IV-9 pour préciser comment se déroule le processus d'édition.

Initialement, lors de l'ouverture d'un document, le gestionnaire de fichier construit le document de référence. Une fois cette entité construite, les différentes vues désirées sont ouvertes. Lors de cette ouverture le gestionnaire de vues, grâce au mécanisme de synchronisation fourni par les classes document met en place la synchronisation entre les différentes vues et le document de référence.

Par la suite, toute opération de l'auteur se fait au travers d'une vue. Cette opération sera directement effectuée sur le document de référence. Celui-ci vérifiera la cohérence de l'opération et formatera ensuite le document pour prendre en compte l'opération d'édition réalisée. Les structures de *document étendu* seront ensuite informées que le document de référence a été modifié. Si la modification se limite aux attributs, les différentes vues étendues seront directement mises à jour, dans le cas d'une modification de structure, elles seront informées de cette modification.

Bien entendu, une vue peut se désynchroniser du document de référence. Dans ce cas, l'auteur aura le choix au moment de la resynchronisation soit d'appliquer toutes les modifications dans le document de référence, soit de reconstruire sa vue à partir du document de référence.

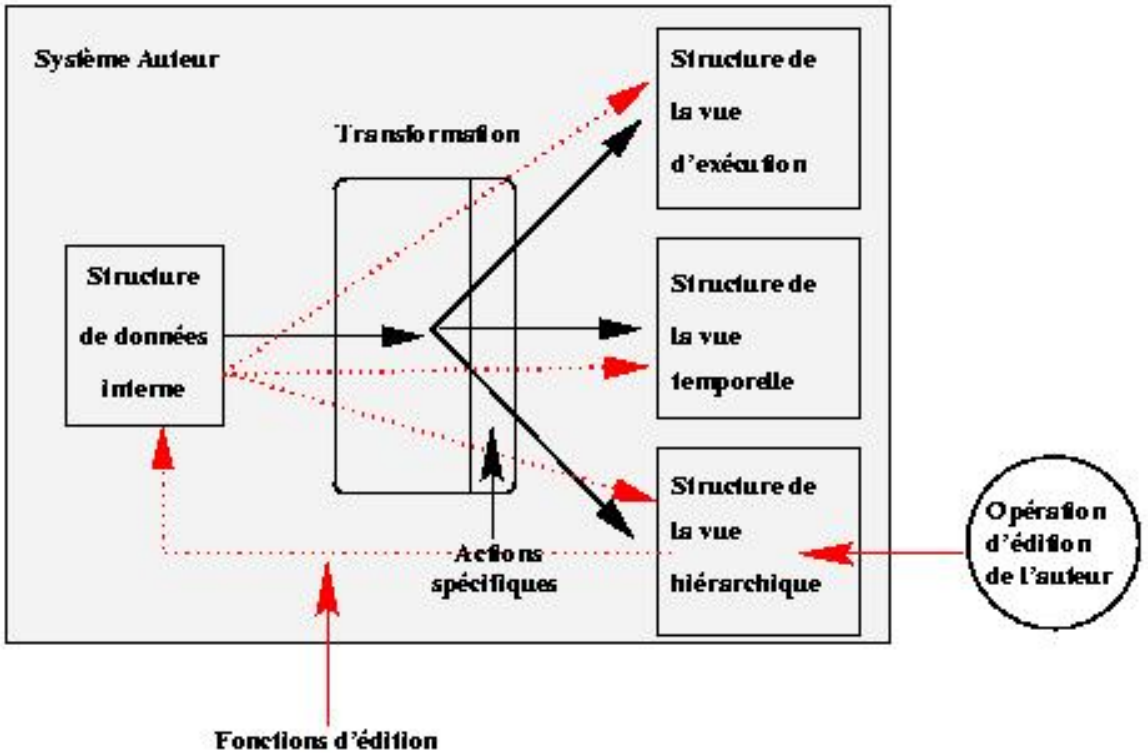


Figure IV-26 : Mécanisme d'édition dans les vues

9.2 Mécanisme de création d'une vue

Il existe dans Kaomi un ensemble de vues prédéfinies. Lors de la réalisation d'un environnement auteur il se peut que le concepteur ait besoin de créer une nouvelle vue ou d'étendre une vue existante. Ces deux mécanismes d'extension ont été prévus dans Kaomi.

La création d'une vue est facilitée par la présence du gestionnaire de vues et par le fait que toute vue ne communique jamais directement avec une autre vue. Tout communication passe par le gestionnaire de vues.

De même toutes les opérations d'édition passent par le document de référence et sont répercutées sur les documents étendus par un mécanisme de synchronisation directe des structures de données.

9.3 La vuede présentation

La vue de présentation (Figure IV-27) est celle qui implémente les services de présentation du document, et qui fournit des services tels que joue / stop / pause / reprend. Cette vue permet aussi d'éditer le document. Elle permet par exemple de modifier les attributs spatiaux des objets par des

manipulations directes.

Dans la Figure IV-27, on peut voir les trois zones graphiques qui composent la vue d'exécution :

- *Le menu*, qui permet à l'auteur d'accéder aux différentes fonctions de l'application.
- *La zone de présentation*, dans laquelle le système joue le document et qui permet aussi de visualiser et de percevoir les relations spatiales.
- *La zone de contrôle*, qui permet à l'auteur de contrôler l'exécution du document.

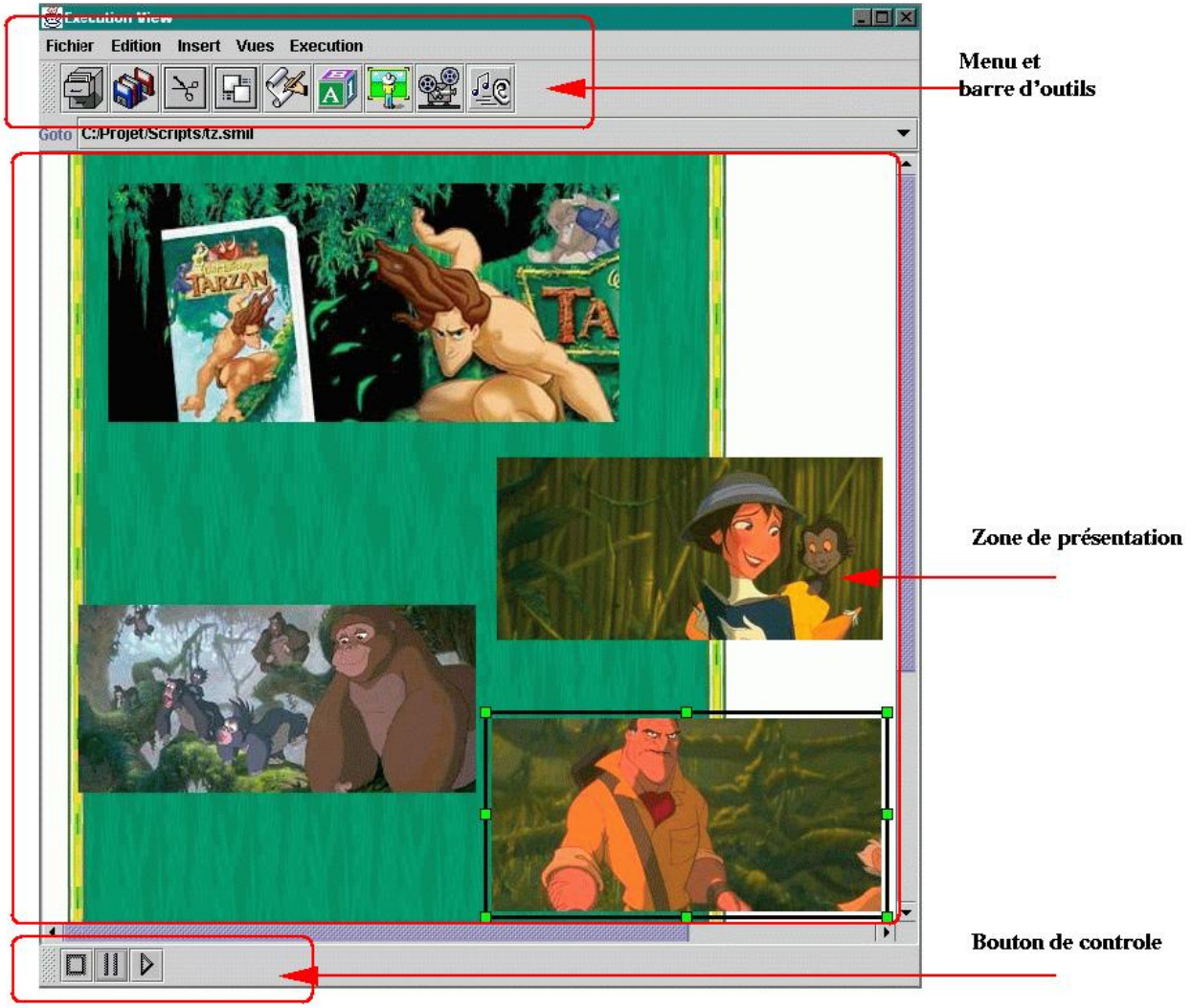


Figure IV-27 : Vue de présentation offerte par Kaomi dans SMIL-Editeur

La vue de présentation se distingue des autres vues par la présence d'un gestionnaire d'exécution ([Layaïda97], [Sabri99]), qui permet de jouer le document. Le gestionnaire de présentation se base sur une structure de graphe (voir section 7.3) et utilise les médiateurs (*players*) fournis dans Kaomi pour présenter les différents médias, un fichier de ressource permet de lier le type du média au médiateur utilisé pour le jouer. Les médiateurs sont des classes qui permettent de visualiser des médias dans une vue, ils sont utilisés dans la vue de présentation, la vue temporelle, et la vue vidéo structurée. Ces médiateurs sont construits en utilisant les JMF. L'extension du gestionnaire de présentation pour l'intégration de nouveaux médias est facilitée du fait que le gestionnaire de présentation fait abstraction du média qu'il manipule grâce à la définition d'interface et au mécanisme d'héritage [Sabry99].

Lors de la création de la structure de document étendue dans la vue de présentation, Kaomi associe à chaque élément temporel élémentaire un médiateur qui gérera la présentation de cet élément en utilisant les informations de présentation du médiaElement associé au noeud temporel.

A chaque élément composite sera associé un gestionnaire de présentation qui utilisera le graphe temporel pour présenter les fils de l'élément composite.

Les gestionnaires de présentation héritent de la classe médiateur, ce qui permet de traiter de manière transparente l'exécution d'objets élémentaires et d'objets composites dans le gestionnaire de présentation.

La vue de présentation utilise des gestionnaires spatiaux pour maintenir les relations lors des manipulations de l'auteur.

9.4 La vue hiérarchique

La vue hiérarchique (Figure IV-28) permet de visualiser les structures hiérarchiques (temporelle et spatiale) du document. Cette vue peut aussi servir à visualiser d'autres structures comme celle de la vidéo. Cette vue fournit les services classiques d'une vue hiérarchique, comme l'ouverture et la fermeture de noeuds, l'insertion et l'ajout de noeud,...

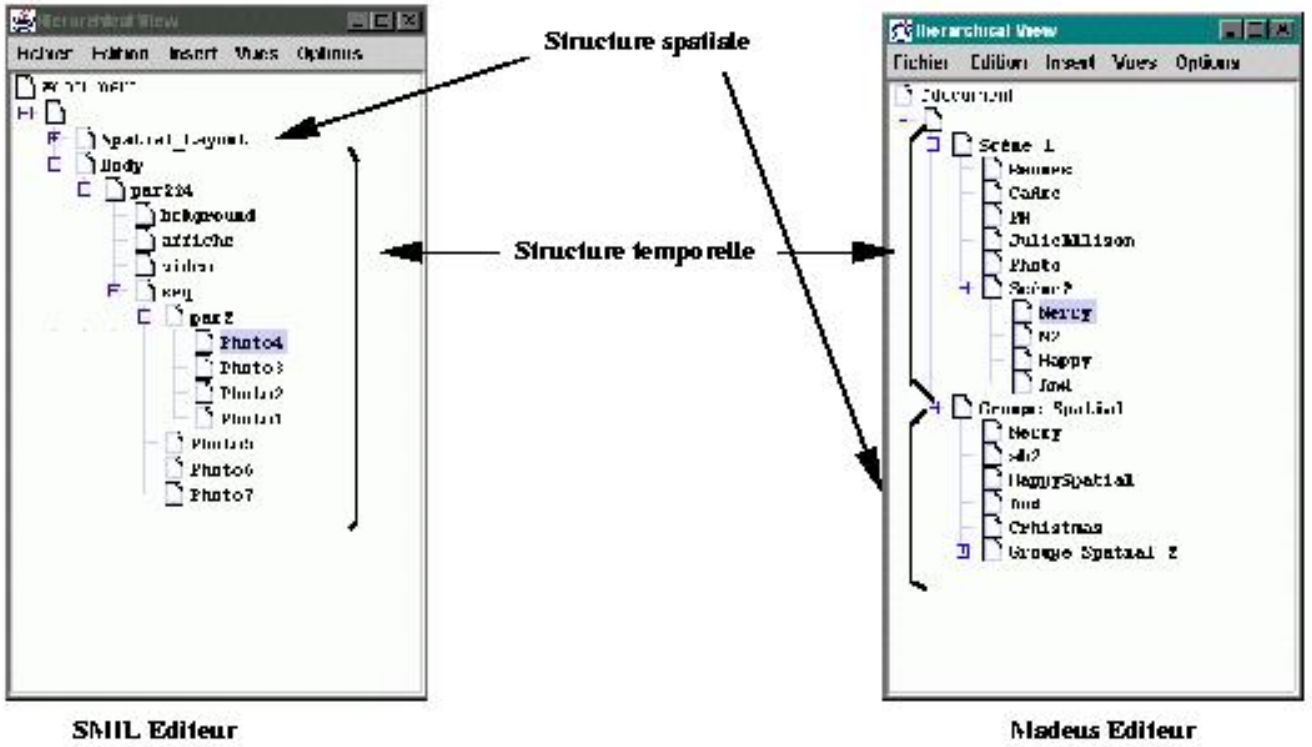


Figure IV-28 : Vue hiérarchique

Lors de la création de cette vue seules les informations hiérarchiques sont copiées. Certains attributs peuvent être copiés en plus si l'auteur désire par exemple filtrer l'affichage des noeuds selon la valeur d'un attribut.

9.5 La vue résumé

La vue résumé (Figure IV-29) visualise les instants clés de la présentation, on peut rapprocher cette vue des vues qui présentent les images clés dans les systèmes de gestion de vidéo. Ces instants clés peuvent être soit calculés automatiquement par le système soit définis par l'auteur. La capture des différents instants clés se fait à la demande de l'auteur dans la vue de présentation. Les différents instants clés sont stockés sous forme d'images qui seront utilisées lors de l'ouverture de la vue de résumé. Dans le cas où le système calcule les instants clés, il peut, par exemple, soit faire une coupe de la présentation toute les 10 secondes, soit calculer les instants de début et de fin des objets, et réaliser une coupe à chaque instant où un objet commence ou se termine. Cette vue permet d'avoir une vision plus globale de l'exécution du document par rapport à la vue de présentation, car on visualise temporellement plusieurs vues spatiales du document.

Figure IV-29 : Vue résumé

Cette vue se distingue des autres dans le sens qu'elle utilise explicitement d'autres vues. Lors de l'ouverture d'une vue résumé, celle-ci construit une structure de données de référence à partir des différentes images du résumé. De ce fait, cette vue peut utiliser plusieurs présentations pour le jeu d'images dont elle dispose, pour cela il lui suffit de placer des relations spatiales et temporelles entre les différentes images. Le résumé est donc construit comme un document multimédia. Une fois cette structure de données construite, elle demande l'ouverture d'une vue de présentation utilisant sa structure de données comme document de référence.
Ainsi, l'auteur ne voit jamais réellement de fenêtre résumé, il ne manipule en fait qu'une fenêtre de présentation.

9.6 La vue attributs

La vue attributs (Figure IV-30) permet à l'auteur de modifier les attributs de l'objet sélectionné. Cette vue, qui évolue en fonction du type d'objet manipulé, permet de définir tous les attributs de l'objet. C'est une palette qui est construite de manière générique à partir d'un fichier de ressources (voir section 5.1.3), elle est donc facilement adaptable pour chaque format de document.

Dans cette vue, les attributs sont regroupés par familles, ces familles sont indépendantes de la structure de données, c'est-à-dire que, par exemple dans une même famille, on peut retrouver des attributs temporels mais aussi des attributs liés à la présentation de l'objet, c'est le cas par exemple de l'attribut *Fill* en SMIL.

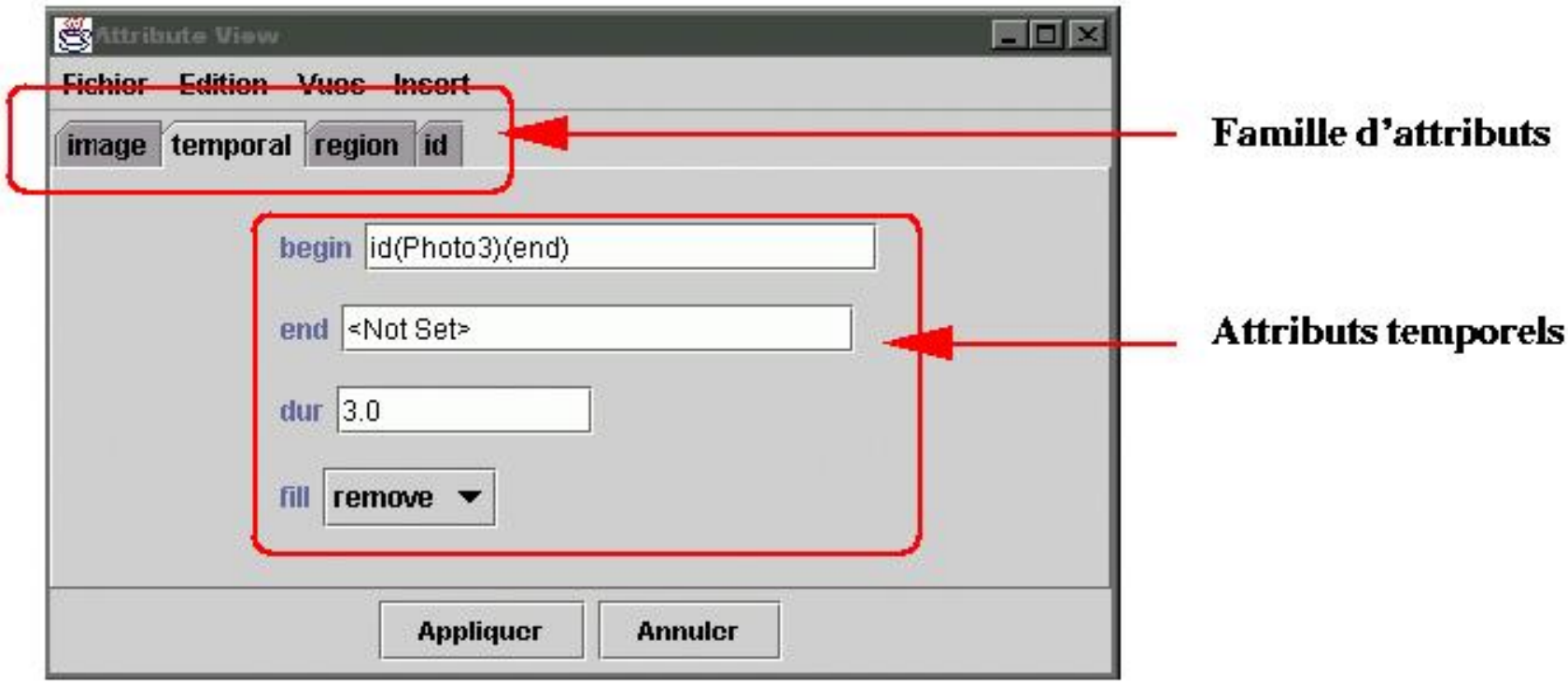


Figure IV-30 : Visualisation des attributs d'un objet

9.7 La vue temporelle

La vue temporelle (Figure IV-31) implémente les services décrits dans la section 3.2.1 du chapitre III. Elle permet de représenter plusieurs informations : attributs, relations, espace de solutions.

Cette visualisation se fait selon deux modes définis ci-dessous :

Statique :

- *Les attributs temporels* : de par le placement sur un axe temporel des objets et leur dimension proportionnelle à leur durée, l'auteur perçoit directement la valeur des attributs temporels des objets (début, fin, durée). Cette visualisation explicite de ces attributs permet à l'auteur d'avoir une vue globale de l'exécution temporelle. On peut noter que cette vue est temporellement honnête [Tufte83], puisqu'on peut tracer un axe temporel absolu en même temps.
- *Les relations temporelles* : les délais introduits par les relations sont visualisés de manière explicite (rectangle jaune), permettant à l'auteur de percevoir l'espace de solutions.
    - *La structure temporelle* : par un mécanisme d'englobement de boîtes, on représente explicitement les différents niveaux de la hiérarchie

temporelle. L'auteur peut ouvrir/fermer ces niveaux à volonté. Cela permet d'alléger l'affichage dans le cas des gros documents ainsi que de percevoir l'enchaînement des objets composites.

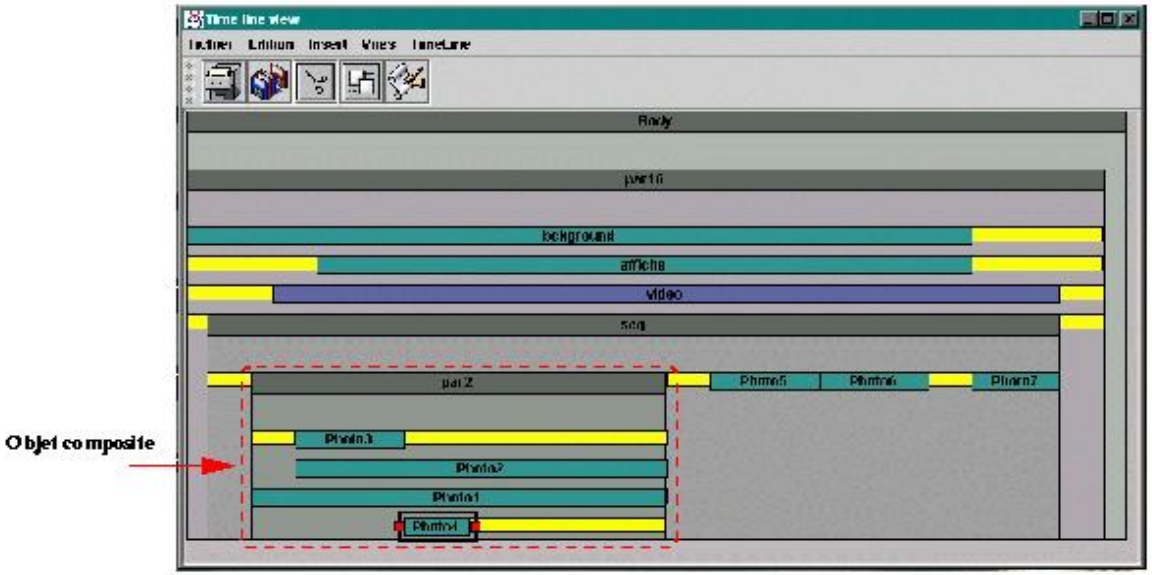


Figure IV-31 : Vue temporelle

Opérationnel :

- *Les informations d'édition* : le mécanisme d'édition directe dans la vue temporelle qui est réalisé en grande partie à l'aide de résolveurs de contraintes sera illustré dans le chapitre VI avec la présentation de l'environnement Madeus-Editeur.
- *La visualisation d'un rapport d'exécution*: la vue temporelle permet aussi de visualiser le rapport d'une exécution (Figure IV-32). Dans cet exemple, deux objets ont été interrompus lors de la présentation (la durée du composite était trop courte), et un objet n'a pas été joué. L'intérêt d'une telle vue au cours du processus d'édition est de permettre à l'auteur d'avoir une explication de l'exécution de son document, en lui permettant par exemple de comprendre pourquoi tel ou tel objet n'a pas été joué.

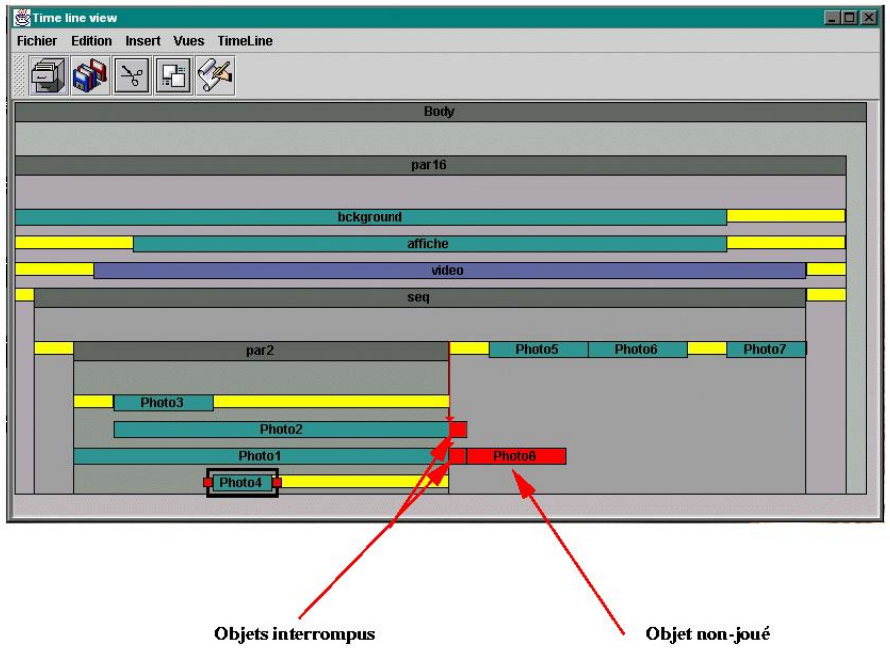


Figure IV-32 : Vue temporelle présentant un rapport d'exécution dans Madeus-Editeur

Nous allons maintenant décrire le mécanisme de construction de la vue temporelle.

A la création d'une vue temporelle, seule la structure temporelle du document de référence est copiée. La vue temporelle utilise le mécanisme des médiateurs de la vue de présentation pour afficher une représentation graphique des noeuds temporels dans la vue de présentation. De ce fait, la vue temporelle crée une structure spatiale (attributs spatiaux sur chaque élément) qui servira aux médiateurs pour afficher les noeuds temporels dans la vue temporelle.

Dans le cadre des objets composites la vue temporelle crée un nouveau médiateur qui a deux représentations graphiques permettant ainsi à la vue temporelle d'ouvrir/fermer les noeuds composites.

La vue temporelle utilisera un gestionnaire de placement pour calculer le placement des objets dans la vue et pour répercuter les modifications de l'auteur (Chapitre V section 7).

La création de la vue rapport d'exécution a été réalisée de manière à minimiser le temps de création de cette vue. Pour cela nous importons un rapport d'exécution dans la vue temporelle. Le rapport d'exécution n'est qu'une exécution particulière du document. De ce fait, il contient juste les durées réelles des objets et l'explication de leur fin (fin normale, interruption par un autre objet,..). La vue temporelle doit donc juste mettre à jour la durée des objets et utiliser les informations complémentaires pour aider l'auteur dans la compréhension de l'exécution.

9.8 Bilan des vues

Le mécanisme de création des vues ainsi que l'implémentation d'une classe vue a permis de faciliter la création de nouvelles vues. En effet, cette classe a permis de mettre en commun tous les mécanismes de synchronisation entre les vues ainsi qu'avec le document de référence.

Les différentes vues que nous avons présentées permettent d'offrir à l'auteur une visualisation des différentes informations contenues dans son document, et les vues de navigation lui permettent de se déplacer facilement dans l'espace de solutions défini par son document. Ces différentes vues permettent de répondre aux attentes et besoins des auteurs définis dans le chapitre II (section 2.2).

En particulier, les différents besoins de visualisation et d'édition par manipulation directe des dimensions spatiale et temporelle sont satisfaits (Chapitre III section 3).

L'édition est facilitée du fait que l'auteur a à sa disposition un ensemble de vues, de ce fait, il peut utiliser la vue la plus adaptée à son opération d'édition. Notons qu'une des difficultés pour l'auteur peut être le passage entre les différentes vues. Ce passage est facilité dans Kaomi grâce à la synchronisation des vues. Cette synchronisation se fait à la fois sur les objets sélectionnés mais aussi sur l'instant de présentation (vue temporelle/vue d'exécution).

10 Apport de Kaomi par rapport aux autres boîtes à outils

Il existe aujourd'hui de nombreuses boîtes à outils dans le domaine du multimédia et des documents. On peut classer les boîtes à outils en cinq catégories :

- Les boîtes à outils qui permettent de visualiser les médias. C'est par exemple le cas de Java Media Framework [JMF00] de SUN qui permet de définir une interface de manipulation des différents médias dynamiques ou statiques. Des boîtes à outils comme Nsync [Bailey98] permettent de gérer le placement temporel des objets. Ces boîtes à outils sont de bas niveaux et certaines comme les JMF sont d'ailleurs utilisées par Kaomi.
- Les boîtes à outils qui permettent de construire des applications graphiques interactives. On peut citer Amulette [Myers96] et Toolbook [Asymetrix99]. Ces dernières offrent un haut niveau d'abstraction pour définir l'interface et le comportement visuel de l'application. De nombreux mécanismes de ces boîtes à outils ont été repris dans des bibliothèques telles que les Swing ou les JMF.
- Les boîtes à outils qui permettent de développer des outils spécialisés. Ces dernières sont une spécialisation des précédentes. Par exemple, l'environnement Melissa [Pernin96] construit au dessus de Toolbook fournit un environnement ouvert est adaptable qui permet à un auteur n'ayant pas de compétence informatique de créer des simulations pédagogiques interactives. L'environnement Melissa se charge de générer les composants graphiques dans un langage de scripts. Ce principe est réutilisé dans les différents environnements auteur que nous présenterons dans le chapitre VI.
- Les boîtes à outils qui permettent de concevoir des environnements auteur pour les documents structurés statiques telles que Thot [Quint99]. Cette boîte à outils offre les moyens de construire facilement des environnements en mettant en commun un ensemble de services d'édition et d'aides à l'auteur. Thot ne fournit cependant que des mécanismes pour une édition de documents non temporisés.

- Les environnements auteur extensibles comme MAVA [Hauser00] qui permet à l'auteur de documents multimédias d'étendre le pouvoir d'expression de son système auteur. C'est de cette approche que nous sommes la plus proche. MAVA est réalisée à l'université de Stuttgart. Le modèle temporel de MAVA est basé complètement sur un arbre d'opérateurs ce qui restreint le pouvoir d'expression de l'auteur. Cependant, la boîte à outils MAVA offre à l'auteur le moyen d'étendre ce langage. L'extension se fait au moyen de modules Java que l'auteur développe. Chaque module, respectant une certaine interface, fournit le nouvel opérateur et la sémantique de cet opérateur. L'avantage de cette approche est que la sémantique de l'opérateur est donnée dans le module. On peut noter, d'une part que l'auteur pour étendre son langage, doit avoir des connaissances en programmation Java, et d'autre part que si l'auteur désire partager avec d'autres personnes son document, il doit fournir les modules d'extension de son environnement.

Kaomi est une boîte à outils permettant de concevoir des environnements auteur extensibles puisque les environnements auteur peuvent être étendus par le mécanisme de ressources.

11 Conclusion

Au cours de ce chapitre nous nous sommes attachés à mettre en oeuvre une boîte à outils qui permet de construire les environnements idéaux spécifiés dans le chapitre III. Nous avons présenté Kaomi, une boîte à outils qui permet de satisfaire les différents besoins des auteurs, que ce soit au niveau visualisation, qu'au niveau facilité d'édition.

Dans les chapitres suivants, nous allons nous intéresser à l'implémentation des différents services d'aide (Chapitre V) et puis à l'utilisation de Kaomi pour la création d'environnements auteur de documents multimédias.

Chapitre V : Les contraintes au coeur de l application

1.Introduction

Au cours du chapitre III, nous avons choisi de fournir un formalisme d'édition à l'auteur qui était une combinaison d'un formalisme relationnel prédictif et du formalisme du langage multimédia utilisé pour stocker le document. Nous avons vu, dans le chapitre IV, que les structures de données de Kaomi ont été réalisées de manière à stocker les informations liées à ces deux types de formalismes.

Dans ce chapitre, nous allons nous intéresser à l'implémentation des services d'aide, et plus particulièrement aux services de cohérence et de formatage pour le formalisme relationnel offert dans Kaomi. Nous verrons dans le chapitre VI, quelles extensions ou améliorations seront nécessaires pour étendre les mécanismes proposés dans ce chapitre de manière à réaliser le formatage et la vérification de cohérence pour un formalisme non relationnel et/ou imprédictif.

Au cours de cette thèse nous avons fait le choix d'étudier des solutions qui s'appuient sur des techniques de résolution de contraintes pour mettre en oeuvre les services de cohérence et de formatage plutôt que de chercher des solutions ad hoc, nous détaillerons cette justification ci-après. L'objectif de ce chapitre est double : trouver un ou plusieurs résolveurs de contraintes adaptés à nos besoins et les intégrer dans la boîte à outils Kaomi.

L'organisation de ce chapitre est la suivante :

Dans une première étape, nous allons justifier notre intérêt pour les résolveurs de contraintes dans Kaomi (section 2) et présenter les différentes manières d'intégrer un résolveur de contraintes à une application (section 3).

Dans une deuxième étape, nous présenterons le mécanisme général de fonctionnement d'un résolveur de contraintes (section 4) pour nous intéresser plus particulièrement au processus de traduction de nos données vers les points d'entrée de ces résolveurs (section 5).

Dans une troisième étape, nous chercherons le résolveur idéal pour notre contexte. Pour cela, nous présenterons les différents types de résolveurs existants en les classant par rapport à leurs mécanismes de résolution (section 6). Nous chercherons ensuite à évaluer ces différents résolveurs tant d'un point de vue qualitatif que quantitatif dans un contexte d'édition. Pour cela, nous présenterons, dans un premier temps, comment s'effectue l'intégration d'un résolveur de contraintes dans Kaomi (section 7) et dans un deuxième temps les résultats tant qualitatifs que quantitatifs de l'évaluation des différents résolveurs (section 8).

Enfin, pour conclure ce chapitre nous ferons un bilan sur l'apport des contraintes dans la réalisation et l'utilisation d'un environnement auteur de documents multimédias.

2. Utilisation des contraintes

Dans toutes tâches de conception où l'on peut séparer la phase de spécification et la phase de réalisation, il peut être intéressant d'utiliser des mécanismes à base de contraintes pour chacune de ces phases. En effet, l'utilisation des techniques à base de contraintes permet :

- *Au niveau de la spécification,*de simplifier le travail de l'utilisateur (auteur) en lui fournissant un modèle de spécification simple (déclaratif) et paramétrable. Cela permet de dégager l'utilisateur de la résolution de sa spécification.
- *Au niveau de la réalisation,*de simplifier le travail du développeur d'une application en lui fournissant des mécanismes automatiques de résolution de la spécification de l'auteur (qu'elle soit spécifiée en terme de contraintes ou pas).

Dans ces deux catégories d'utilisation des travaux probants ont permis de valider l'utilisation de ces techniques dans un contexte applicatif.

La première catégorie d'utilisation concerne la spécification par l'auteur d'une entité (document, interface) à l'aide de contraintes :

- *Spécification du scénario temporel*: l'auteur spécifie un ensemble de relations entre les objets. C'est le cas de la spécification du scénario temporel dans Nsync [Bailey98]. La résolution des contraintes, réalisée par un module externe (Berkeley Continuous Media Toolkit) se fait au début de présentation, et permet calculer le placement temporel de tous les objets.
- *Spécification de pièces de musique*: dans le logiciel de composition musicale appelé Boxes [Beurivé00], l'utilisateur place des sons les uns par rapport aux autres dans le temps. Il peut les organiser hiérarchiquement (pour obtenir des mélodies ou des pièces de musique), mais il peut aussi déclarer des relations temporelles entre ces éléments. Le résolveur Cassowary [Badros98] est utilisé pour résoudre la spécification de l'auteur.
- *Spécification de documents*. Parmi les utilisations des contraintes pour la spécification de documents multimédias on peut citer Anecdote [Harada96], ISIS [Kim95], Madeus [Layaïda97], et les travaux de Saade [Saade97] qui proposent une édition de documents avec des contraintes spatiales et temporelles.
- *Spécification d'interfaces graphiques* : de nombreux outils permettent de spécifier des interfaces via des mécanismes proches des contraintes. Par exemple, le langage Java permet de définir des composants graphiques avec des politiques de placement pour les objets à l'intérieur du composant. C'est le système qui, au moment de l'affichage, s'occupera de calculer les positions absolues des différents objets en résolvant les contraintes spécifiées par l'auteur. De la même façon, le système Scwm [Badros00] permet à l'auteur de spécifier un ensemble de relations entre les différents objets composant l'interface.
- *Le dessin d'objets géométriques* : dans le cas de Cabri II [Laborde95], les contraintes sont utilisées à la fois pour spécifier les relations spatiales entre les objets mais aussi pour maintenir des propriétés géométriques sur les objets et entre les objets, notamment lors des manipulations de l'utilisateur. Charman [Charman94] utilise lui les contraintes pour l'aide à l'aménagement spatial d'objets (définition de plan par exemple).

La deuxième catégorie d'utilisation des technologies à base de contraintes concerne la résolution d'une spécification (qui n'est pas forcement exprimée sous forme de contraintes). Ces techniques permettent au programmeur de l'environnement de se dégager de la résolution, et du maintien des relations en utilisant des résolveurs externes. Parmi les principales utilisations, on peut citer :

- *L'affichage de champs d'étoiles*: dans l'outil StarGen [Hudson95], l'auteur spécifie un ensemble de valeurs à afficher dans des structures de données abstraites. Le système StarGen traduit cet ensemble de valeurs en un ensemble de contraintes qui seront résolues par un résolveur de contraintes externe, pour être finalement affichées à l'écran.
- *Pour la spécification d'animation*: on peut citer les travaux de TRIP [Takahashi95] où l'auteur peut spécifier des représentations graphiques de données arborescentes mais aussi des animations graphiques en 2D et 3D au moyen de langages abstraits qui sont ensuite convertis en un ensemble de contraintes spatiales permettant de définir l'affichage des objets à l'écran. Cet ensemble de contraintes est ensuite résolu par le résolveur Detail [Hosobe98].

Dans le cadre de l'édition de documents multimédias nous avons vu qu'il était intéressant de séparer la spécification du comportement temporel et spatial du document de la phase de calcul du placement absolu des objets. Des expériences positives ont eu lieu au sein du projet Opéra ([Carcone97] et [Carcone97b]) pour

calculer le placement spatial d'objet. Au cours de cette thèse nous avons eu la volonté d'apporter une réponse globale au problème de formatage et de vérification de cohérence dans le contexte de l'édition de documents multimédias.

Dans Kaomi, les résolveurs sont utilisés pour réaliser les fonctions suivantes :

- Vérification de l'existence de solution, et calcul de l'espace de solutions.
- Propagation des modifications de l'auteur lors de l'édition.
- Maintien des relations et de la cohérence de l'affichage lors des manipulations de l'auteur (formatage lors des déplacements d'objets dans la vue temporelle ou de présentation).

Ces différentes fonctions interviennent principalement dans trois modules suivants de Kaomi (Figure V-1) :

- Le module de visualisation, qui permet par exemple de visualiser l'espace de solutions.
- Le module d'édition, qui après chaque opération d'édition de l'auteur doit assurer la cohérence et le formatage du document.
- Le module d'aide qui au cours des manipulations de l'auteur dans les vues temporelle ou de présentation doit calculer un nouveau placement pour les objets.

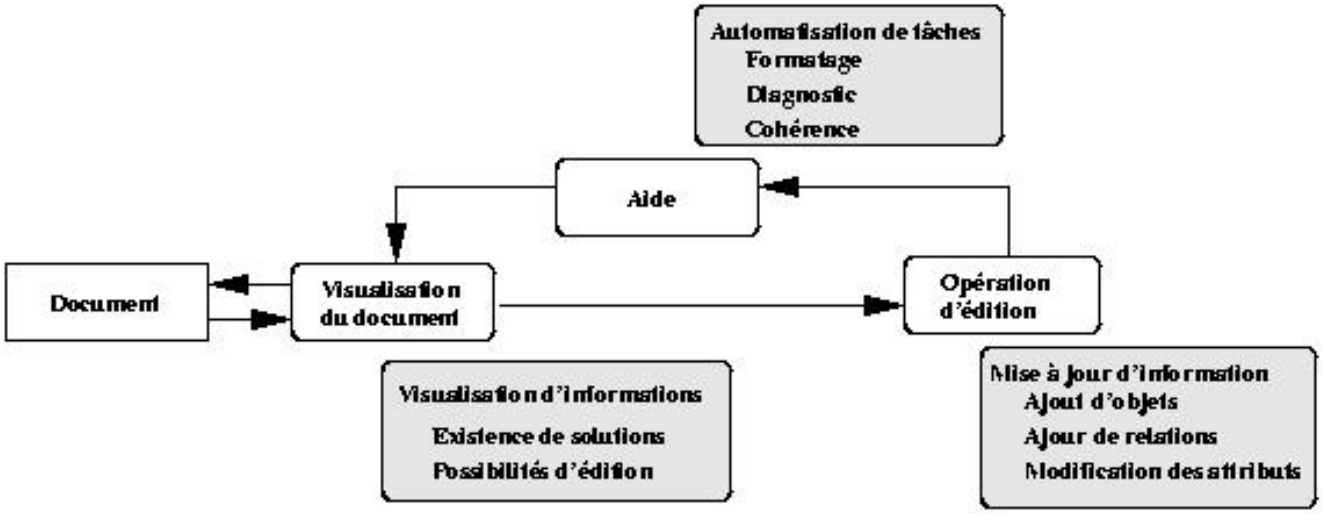


Figure V-1 : Les contraintes dans l'édition

3. Les différentes approches pour choisir ou créer un résolveur

Il existe de nombreuses manières d'aborder la résolution de contraintes :

- *Les langages de construction de résolveurs :* dans ces approches l'utilisateur décrit (programme) dans un langage spécialisé le résolveur de contraintes qu'il désire obtenir. Cette approche est illustrée par Laure [Caseau94] et Claire [Caseau96]. L'utilisateur doit intégrer des mécanismes de bas niveau pour créer un résolveur, le langage lui fournissant par exemple le moyen de stocker l'état courant des variables. Il peut, par exemple, définir ainsi une contrainte qui vérifie que pour chaque élément *x,* la date minimale de début de *x* et inférieure à la date maximale de début de *x*.

```
[define checkconstraint constraint
```

```
for_all x:integer, y:integer,
if [exist z, atleast(z) =x , atmost(z) = y]
check x <= y ]
```

Cette approche nécessite une bonne connaissance des mécanismes de résolutions de contraintes pour réaliser un bon résolveur. Cependant les résolveurs produits sont spécifiques pour un problème particulier et efficaces.

- *Les librairies*: ces librairies fournissent un ensemble de contraintes génériques ainsi que le mécanisme de résolution adapté qui peuvent être utilisés dans un large éventail de situations. L'utilisateur de telles librairies doit traduire son problème dans le formalisme de contraintes offert par la librairie. Cette traduction, selon le problème et la librairie peut être une tâche difficile. Les contraintes globales de CHIP [Beldiceanu94, Aggoun93] sont une illustration de telles librairies. Les résolveurs sont génériques et en général simples d'utilisation, cependant leur efficacité fluctue en fonction du domaine d'application.
- *Les librairies extensibles*: ces librairies fournissent les services d'une librairie de contraintes, mais permettent aussi à l'utilisateur de les spécialiser pour un domaine particulier. L'intérêt de telles approches est qu'elles permettent immédiatement au programmeur d'utiliser le résolveur, tout en lui permettant de l'étendre ou de le spécialiser pour son utilisation si le besoin s'en fait sentir [Badros98].

Dans un contexte d'édition de documents multimédias, nous n'avons pas une spécification du problème unique que nous résolvons en changeant seulement les valeurs possibles des variables. Nous avons une spécification du problème différente à chaque étape du processus d'édition (chacune des spécifications étant proche de la précédente). De ce fait nous avons choisi d'étudier plus précisément les deux dernières approches qui semblent plus flexibles en permettant plus facilement le changement incrémental de la spécification du problème.

4. Mécanisme général de résolution de contraintes

Un résolveur de contraintes est un programme qui à partir d'un ensemble de contraintes calcule une solution (si elle existe) satisfaisant toutes les contraintes.

Nous allons donc dans un premier temps présenter les CSP (Constraints Satisfaction Problem) qui permettent de représenter l'ensemble de contraintes manipulé par les résolveurs de contraintes, et dans un deuxième temps nous définirons la notion de solution d'un CSP.

Un problème de satisfaction de contraintes est la donnée d'un ensemble de variables dont les valeurs sont restreintes par des contraintes. Nous allons nous intéresser exclusivement aux CSP binaires, c'est-à-dire aux CSP ne manipulant que des contraintes entre deux variables.

Le CSP à considérer est un triplé G={X, D, C}, où :

- $X = \{X_1, , X_n\}$ est un ensemble de variables,

- $D = D_1 \times \times D_n$ représente les domaines de ces variables (ensembles de valeurs pouvant leur être affectées a priori),

- $C = \{C_{ij} / 1 <= i,j <= n\}$ est un ensemble de contraintes d'entrées.

Une contrainte C_{ij} exprimera la contrainte entre les variables X_i et X_j, restreignant du même coup les valeurs pouvant être prises simultanément par ces deux variables. Dans le cas fini, elle s'exprimera généralement sous la forme d'un ensemble de couples de valeurs permises.

$C_{ij} = \{ (X_{i1}, X_{j1}), , (X_{ip}, X_{jp}) \}$ sur l'espace $D_i \times D_j$.

Dans le cas des domaines infinis, la contrainte est donnée en intention par un prédicat sur les variables de la contrainte.

On définit ensuite **une solution d'un CSP** comme étant une instanciation de toutes les variables satisfaisant l'ensemble des contraintes, c'est-à-dire :

une solution est un ensemble $\Delta = \{X_1$ dans $D_1, , X_n$ dans D_n / pour tout C_{ij} dans C on a (X_i, X_j) dans $C_{ij}\}$.

On dit qu'un système est sous contraint lorsque l'ensemble des contraintes n'est pas suffisant pour définir une solution unique au CSP.

L'expérience nous montrera plus tard que dans un système auteur nous avons de tels systèmes du fait qu'un auteur ne spécifie pas un ensemble de contraintes suffisant pour cela. Notons que c'est ce qui fait la force d'un environnement auteur à base de contraintes, l'auteur ne spécifie que partiellement son document laissant la tâche de calculer le placement spatial et temporel des objets à l'environnement auteur.

Le système de résolution a donc le choix parmi un ensemble de solutions. Se pose alors le problème de la pertinence de la solution choisie par rapport à la spécification de l'auteur. De ce fait, il est nécessaire de donner des informations supplémentaires au résolveur pour orienter le calcul d'une solution. Ces informations peuvent être assimilées à des préférences de l'auteur ou du système auteur vis-à-vis du résolveur pour l'aider dans son choix de la solution.

Ces préférences peuvent être données sous trois formes (voir Figure V-2):

- Complément d'information sur les données d'entrée, avec par exemple, l'ajout de contraintes ou de priorités sur les contraintes pour orienter la recherche de solutions.
- Contrôle sur le mécanisme de résolution, avec par exemple la définition d'heuristiques sur la méthode de résolution.
- Contrôle sur les solutions produites, avec par exemple, un choix d'une solution parmi un ensemble de solutions trouvées.

Nous verrons plus précisément ces techniques dans la section 5.3.

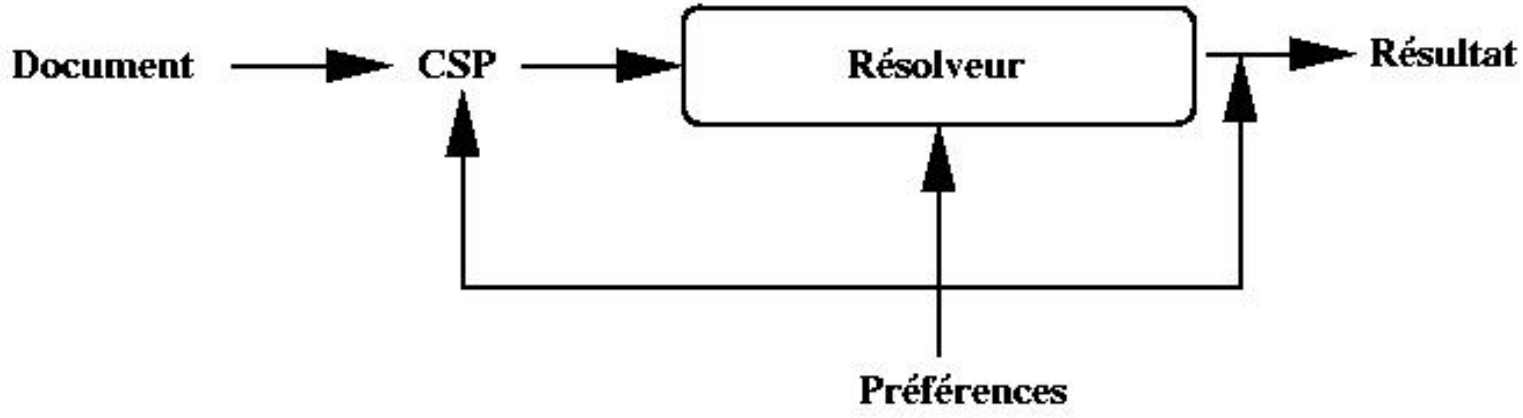


Figure V-2 : Orientation du processus de résolution

5. Traduction de notre problème vers un CSP

L'objectif de cette section est de savoir quelles sont les contraintes qui doivent être données en entrée du résolveur. Pour cela, nous avons besoin d'une part de voir quelles informations du document doivent se traduire en contraintes (section 5.1), et d'autre part analyser l'influence du cycle d'édition sur le mécanisme de

résolution (section 5.2). Nous verrons comment cette influence se traduira vis-à-vis du résolveur (section 5.3).

5.1 Traduction des informations contenues dans le document vers un CSP

Dans le cadre de l'édition de documents multimédias nous avons deux éléments à modéliser :

- Définition des attributs des médias (X, Y, Début, Fin, ..) en variables ;
- Traduction des relations et des événements en contraintes.

Au cours de cette section, les propositions que nous allons faire s'appliquent aux dimensions temporelle et spatiale. Nous illustrerons indifféremment chacun des aspects soit à l'aide d'un exemple temporel, soit d'un exemple spatial.

5.1.1 Expression des attributs temporels et spatiaux dans un CSP

Dans le cas d'un scénario temporel exprimé sous forme de contraintes, chaque objet (ou élémentTemporel) est défini par trois variables (Début, Fin et Durée). Le domaine de ces variables est $[0, +\infty [\cup [?]$. La valeur [?] indique que la variable n'est pas affectée. Chaque variable flexible introduit une contrainte correspondante à son intervalle de durées possibles ($BorneInf \leq Variable \leq BorneSup$). Une contrainte supplémentaire permet de conserver la sémantique des variables (début + durée = fin).

Dans le cadre du scénario spatial, chaque objet est défini par six variables : Haut, Bas, Gauche, Droite, Largeur, Hauteur, chacune étant définie dans un intervalle de valeur ($[0, +\infty [\cup [?]$). Ainsi, chaque variable flexible introduit une contrainte correspondante à son intervalle de valeurs possibles ($BorneInf \leq Variable \leq BorneSup$). De plus, deux contraintes entre les variables (Gauche + Largeur = Droite et Bas + Hauteur = Haut) expriment leur sémantique.

5.1.2 Traduction des relations en terme de contraintes

Comme nous l'avons montré dans le chapitre IV (section 4.4.3), les relations temporelles et spatiales sont converties en rélems. La conversion de relations en rélems se fait à chaque ajout ou retrait de relations. Nous allons donc nous intéresser maintenant au processus de conversion de ces rélems en terme de contraintes.

Les rélems sont directement exploitables par les résolveurs de contraintes. En effet, chacun des rélems ($<, =, \Delta$) se traduit directement en termes de contraintes.

Dans l'exemple de la Figure V-3, on peut voir des exemples de telles conversions. La principale difficulté vient des relations "$\Rightarrow$ " et "$\Leftarrow$ " qui ont un comportement assimilable à un comportement d'interruption. Nous rappelons que Début A $\Rightarrow$ Début B signifie : si A est joué, alors le début de A déclenchera le début de B. Comme nous sommes dans un contexte déterministe, nous pouvons savoir statiquement si l'objet A est joué ou non. Pour cela il suffit de résoudre le système, si A est joué on rajoute la contrainte A.Début = B.Début.

Nous avons choisi d'insérer une nouvelle variable FinEffective, en plus des trois variables temporelles précédentes, pour représenter les valeurs prises lors de l'exécution, valeurs qui sont potentiellement différentes des valeurs choisies statiquement par le formateur à cause des relations d'interruption.

Ce choix, permettra par exemple dans la vue temporelle de représenter explicitement la valeur prévue statiquement par le résolveur, et de visualiser en même temps la valeur réelle prise par le média du fait de l'interruption (rapport d'exécution).

Rélem	Contraintes

Fin A ≥ Début B	C_1:= A.Fin ≥ B.Début
Début A ≤ Début B	C_2:= A.Début ≤ B.Début
Début A = Début B	C_3:= A.Début = B.Début
Début A = Début B + t	C_3:= A.Début = B.Début + t
Début A ⇒ Début B	Si A est joué : C_5:= A.Début = B.Début
Début A ⇐ Fin B	Si B est joué : C_6:= B.FinEffective = A.Début B.Fin > B.FinEffective
Fin A ⇐ Fin B	Si B est joué : A.FinEffective = B.FinEffective et A.FinEffective < A.Fin

Figure V-3 : Conversion rélem-contraintes

L'ensemble des variables définies dans le document est introduit dans le résolveur, ainsi que les contraintes déduites des rélems.

Comme pour les relations et les rélems, nous stockons les contraintes dans le système et, plus particulièrement, dans le résolveur. Nous gardons un lien pour chaque relation vers les rélems qu'elle a engendrés. Nous gardons aussi pour chaque rélem la liste des contraintes qu'il a créés. De cette manière, à chaque retrait de relation nous connaissons les rélems et les contraintes à supprimer. Il est également très facile dans le cas d'une contrainte insatisfaite de retrouver la relation qui l'a induite.

5.1.3 Traduction des événements en contraintes

Dans le chapitre IV (section 4.4.6), nous avons vu comment traduire les différents types d'événement dans notre formalisme d'édition. On peut noter que dans le cas où les objets sont prédictifs et tous les événements prédictifs alors ils sont traduisibles en termes de rélem. Or nous sommes dans le formalisme relationnel prédictif offert par Kaomi. Les événements sont donc traduits sous la forme des rélems "⇒ " et "⇐ ", et introduits dans le résolveur de contraintes.

Dans le cas où nous pouvons évaluer statiquement la valeur des variables correspondante à la liste des dates d'occurrences d'un événement, l'intégration des événements dans le résolveur se fait facilement. Il suffit en effet de transformer chacune des instances d'occurrence de l'événement en une nouvelle variable du résolveur.

Par exemple, l'événement :

```
Source = B
```

```
Destination = A
NbOccurence = 1
Date d'occurrence= {t tel que B se termine}
Conditions = { }
Actions = {A.jouer }
```
sera traduit par le rélem : B.Fin ⇒ A.Fin

et donc par les contraintes :

A.FinEffective = B.FinEffective et A.FinEffective < A.Fin

5.2 Influence du cycle d'édition sur le mécanisme de résolution

Nous avons soulevé dans les chapitres II (section 2.2) et V (section 4) la nécessité de contrôler le choix de la solution en cas de problèmes sous contraints. Dans notre contexte d'édition, celle-ci engendre des besoins précis que doivent satisfaire les résolveurs de contraintes :

- *Prise en compte de préférences de l'auteur* : l'auteur d'un document multimédia intervient de manière directe dans le formatage de son document. Par exemple, il peut vouloir spécifier une valeur préférée sur ses médias. Le système de contraintes devra permettre de prendre en compte de telles préférences qui sont explicitement demandées par l'auteur.

Par exemple, les trois politiques de formatages suivantes peuvent être pertinente dans un contexte d'édition de documents multimédias :

- Privilégier la déformation des délais par rapport aux autres objets.
- Tenir compte du scénario temporel, par exemple, si l'auteur définit une séquence de deux objets qui dure 10 secondes, et que chacun des objets a un intervalle de durées possibles de [1,10], il est intéressant pour l'auteur que le système de contraintes évalue la durée de ces deux objets à 5 secondes.
- Tenir compte des valeurs intrinsèques des objets. Par exemple si le premier objet est une vidéo avec une durée intrinsèque de 6 secondes et le deuxième une image, alors le résolveur de contraintes évalue le premier objet à 6 secondes et le second à 4 secondes.

On peut voir au travers de ces exemples, que pour un même problème il existe plusieurs solutions de formatage qui peuvent être pertinentes suivant le contexte. L'auteur doit donc aider le système vis-à-vis de ces préférences.

- *Maintien de proximité vis-à-vis de la solution courante*. De par les opérations d'édition de l'auteur, le document est en constante évolution. Cependant, il est important que le système auteur ne change pas complètement à chaque opération d'édition le document (les solutions temporelle et spatiale associées au document). C'est pour cela que le résolveur de contraintes doit être capable de prendre en compte cette caractéristique, et permettre une certaine localité dans le calcul des solutions.
- *Connaissance de*la flexibilité dans le document (ou dans le système de contraintes) **:** comme nous avons pu le voir dans le chapitre III (section 3.3.2), il est important pour l'auteur de connaître la flexibilité du document. L'intervalle de validité d'une variable est l'ensemble des valeurs permises pour cette variable tel qu'il existe une valuation pour les autres variables qui rende le scénario cohérent.

De manière indépendante aux besoins qui viennent d'être cités, notons que dans un contexte d'environnement interactif l'efficacité de la méthode de résolution est un facteur très important. Cette efficacité se traduit généralement par l'utilisation de techniques incrémentales (elles profitent de la solution courante pour calculer la nouvelle solution). Là où un résolveur non-incrémental reconsidérerait l'ensemble des contraintes, même en cas de moindre changement, un résolveur incrémental ne réévalue que les contraintes affectées par la dernière perturbation.

5.3 Recherche de la meilleure solution

Les deux premiers points évoqués ci-dessus montrent bien qu'un besoin essentiel pour un environnement auteur est d'obtenir une solution pertinente (quand elle existe) pour l'ensemble des relations définies par l'auteur. Cette solution est utilisée pour afficher le document dans la vue de présentation et pour la jouer dans le temps. Le document défini par l'auteur peut produire un large ensemble de solutions et le système de contraintes doit en choisir une dans cet ensemble selon le critère de pertinence.

Cela pose dans un premier temps le problème de la définition de ce critère, en effet celui ci dépend du contexte d'édition et dans un deuxième temps cela nécessite l'utilisation d'un résolveur pouvant prendre en compte ce critère dans sa recherche de solution.

Nous avons vu dans la section 5.2 que l'auteur pouvait en fonction du contexte, vouloir exprimer différentes préférences. On voit, au travers de ces exemples, que la notion de bonne solution n'est pas évidente, et qu'elle dépend énormément du contexte et du souhait de l'auteur.

Le système auteur (aidé par l'auteur) doit donc guider le résolveur de contraintes vers une bonne solution.

Il existe deux manières de guider les résolveurs de contraintes. La première consiste en l'utilisation de contraintes hiérarchiques [Borning87], la seconde utilise une heuristique. Le choix entre les deux méthodes dépend du résolveur, dans la section 6, nous préciserons pour chaque résolveur la méthode d'orientation qu'il offre.

Nous venons de voir qu'il était important d'obtenir la meilleure solution. Encore faut-il être capable d'évaluer la qualité d'une solution. Dans ce but, nous allons maintenant définir une fonction d'évaluation, qui va nous permettre de mesurer un critère de qualité d'une solution (section 5.3.1). Nous présenterons ensuite les deux manières d'orienter les résolveurs de contraintes de façon à maximiser ce critère (section 5.3.2 et section 5.3.3).

5.3.1 Définition d'une fonction de distance

Cette fonction de distance servira d'une part à orienter les résolveurs de contraintes et d'autre part lors de l'étude des différents résolveurs de contraintes pour évaluer la qualité des solutions proposées.

Pour répondre au besoin de proximité entre solutions successives on définit la qualité d'une solution S_2, obtenue après une action d'édition de l'auteur appliquée sur une variable V, par sa distance par rapport à la solution précédente (appelée S_1).

Le premier paramètre (appelé P_1) qui peut être pris en compte pour la définition de cette fonction est celui de la distance entre la variable V modifiée par l'auteur et les variables modifiées par le système dans S_2 pour mettre le système dans un état cohérent. On peut donc définir la distance entre deux variables comme le nombre de contraintes entre ces variables dans le graphe de dépendances (Figure V-4). Le graphe de dépendance est construit de la manière suivante :

- Les variables du jeu de contraintes représentent les noeuds.
- Pour toutes les contraintes qui impliquent au moins deux variables, on ajoute un arc entre chaque couple de noeuds représentant les variables de la contrainte.

Par exemple, si nous avons les cinq contraintes ci-dessous représentées par le graphe de dépendance de la Figure V-4 :

$$(A > B + C), (C > D), (B = E + F), (E = G), (G = D)$$

La distance entre les variables A et D est : distance(A, D) = 2.

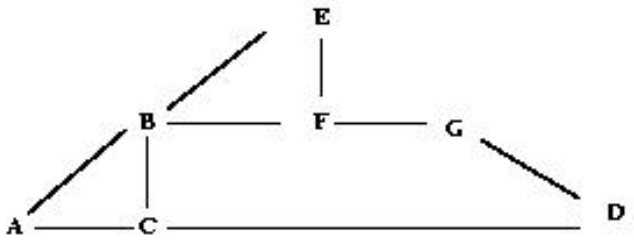


Figure V-4 : Mesure de la distance entre deux variables

Une telle notion de distance dépend évidemment du jeu de relations définies entre les objets. Intuitivement on peut voir que l'objectif de minimiser ce paramètre est de favoriser en priorité des modifications.

D'autres paramètres peuvent être utilisés pour compléter cette notion de distance entre deux solutions :

- P_2 : L'écart type des valeurs des variables entre les deux solutions.
- P_3 : Le nombre d'objets modifiés.
- P_4 : Le nombre d'objets modifiés, pondéré par le type de ces objets. Ce paramètre permet de préférer par exemple des solutions dans lesquelles le changement de durée est effectué sur un texte ou une image plutôt que sur une vidéo ou sur un objet sonore. En effet, on préférera conserver la qualité optimum d'une vidéo ou d'un son.

Finalement la fonction de distance est une combinaison de tous ces paramètres, ce qui signifie que la fonction de distance F_d est définie comme suit :

$$F_d (S_1, S_2) = aP1 + bP2 + cP3 + dP4$$

où a, b, c et d sont les poids associés aux paramètres tels que a + b + c + d = 1.

Cette fonction peut être ajustée (en changeant les valeurs de a, b, c et d) pour satisfaire les besoins de l'environnement auteur. Cette définition est suffisamment générale pour être adaptée au choix de l'auteur ou au contexte d'édition. Nous avons pu le vérifier par exemple dans l'éditeur de Workflow (Chapitre VI, section 3).

Nous allons voir à présent comment orienter la recherche de solution de deux manières différentes.

5.3.2 Utilisation des contraintes hiérarchiques

L'idée de base des contraintes hiérarchiques [Borning92] est d'associer un poids aux contraintes. Lorsque le résolveur de contraintes ne peut satisfaire deux contraintes qui sont en opposition, il satisfait celle de poids le plus fort.

L'utilisation de contraintes hiérarchiques permet entre autres, d'orienter la recherche de solutions. Pour cela, on insert en plus des contraintes du problème initial (de poids le plus fort possible), des contraintes de poids plus faible qui serviront à exprimer un ensemble de préférences.

Par exemple, dans l'exemple de la Figure V-5, la spécification permet d'indiquer au résolveur ce qu'il doit modifier en priorité lors d'une résolution.

```
x=4 poids = Fort

y=3 poids = Faible

x=y poids = Très Fort
```

Figure V-5 : Exemple de spécification utilisant des contraintes hiérarchiques

Aujourd'hui, la plupart des résolveurs utilisés dans Kaomi permettent de définir de tels poids sur les contraintes (Deltablue, Cassowary). Ces contraintes peuvent être utilisées par le système auteur pour donner des informations externes aux résolveurs de contraintes, comme la durée optimum des médias qu'il faut chercher à conserver. Chaque étape d'édition nécessite de ce fait non seulement un ajout/retrait de variables et/ou de contraintes relatives à l'action d'édition de l'auteur, mais aussi une phase de gestion de cet ensemble de contraintes additionnelles.

5.3.3 Utilisation d'heuristiques

Les mécanismes de résolution de contraintes possèdent en général les deux étapes suivantes :

- choix d'une variable à évaluer ;
- choix d'une valeur pour la variable choisie.

Le système peut définir des heuristiques pour orienter le choix de la variable et/ou le choix de la valeur pour la variable.

Par exemple, dans la première étape une fonction peut trier les variables à évaluer en fonction du type des objets (texte, son, vidéo,..), et du nombre de relations que les objets ont directement ou indirectement. La seconde étape peut utiliser une fonction pour permettre de choisir les valeurs des variables en fonction de leur valeur courante et des valeurs Min et Max de l'intervalle.

Cette technique est utilisée par exemple dans Jsolver [HonWai99].

6. Les différentes approches existantes pour le calcul de solution

Nous avons choisi de présenter les résolveurs en fonction du mécanisme de résolution qu'ils proposent. Nous présenterons dans la section 6.1 les méthodes de résolution locales, et dans la section 6.2 les méthodes de résolution globales. Dans une troisième partie (section 6.3) nous présenterons un algorithme basé sur le simplex. Enfin, dans une dernière partie nous présenterons des algorithmes spécialisés que nous avons développés pour la résolution de sous problèmes dans le cadre de l'édition et la présentation de document (section 6.4).

6.1 Les approches locales

Les résolveurs locaux ont pour objectif d'utiliser la connaissance de l'ensemble des contraintes et des dépendances entre elles pour optimiser le calcul de solution après une modification.

Pour cela, ils construisent dans un premier temps un graphe de contraintes (Figure V-6). Chaque variable du problème est représentée par un cercle, chaque contrainte est représentée par un carré, et chaque arc représente l'appartenance d'une variable à une contrainte.

Dans un deuxième temps, à chaque contrainte est associée un ensemble de méthodes qui explicitent comment répercuter la modification d'une variable sur les

autres variables présentes dans la contrainte. Par exemple, une méthode associée à la contrainte C1 (Figure V-6) peut être A ← C - B qui indique comment répercuter une modification de C et/ou B (ce sont les entrées de la méthode) sur A (c'est la sortie de la méthode).

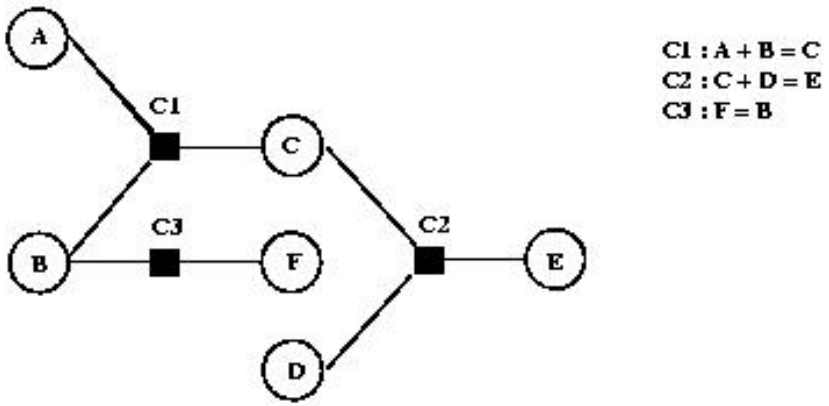


Figure V-6 : Graphe de contraintes

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 :

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

Une des méthodes utilisées pour assurer ces trois fonctionnalités est la méthode par propagation locale de valeur. L'idée sous-jacente à cette technique est de dire que dès que la contrainte possède assez d'informations pour calculer des valeurs, elle les calcule. La phase de résolution se décompose en deux phases : 1. *Une phase de planification* : au cours de celle-ci le résolveur sélectionne une méthode pour chacune des contraintes à recalculer, et uniquement pour celles-ci. Cet ensemble de méthodes est calculé en partant de la variable perturbée, et en identifiant toutes les contraintes potentiellement insatisfaites.

Pour chacune de ces contraintes une méthode de résolution (parmi celles qui sont définies) est choisie. Cette méthode doit prendre la variable perturbée en entrée. Le système doit alors propager les modifications liées à l'utilisation de cette méthode. L'ordre d'exécution correspond à un tri topologique de ce graphe en fonction de son orientation.

Par exemple une orientation possible dans l'exemple de la Figure V-6 est donnée dans la Figure V-7, cette orientation répercute une modification de A sur les variables C, D et F.

2. *Une phase d'exécution* 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.

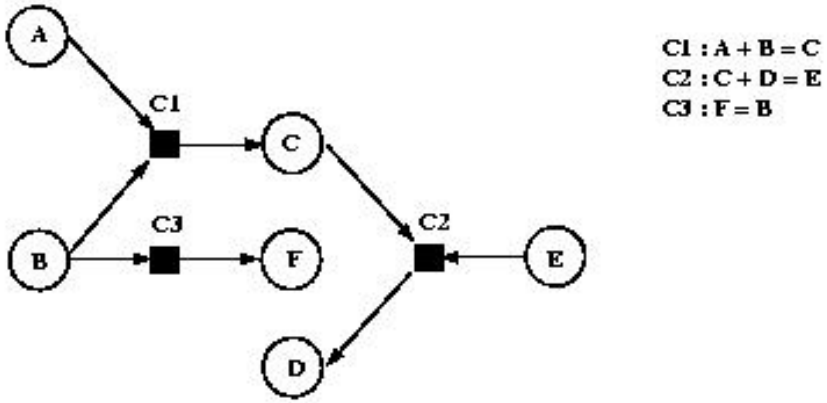


Figure V-7 : Orientation du graphe de contraintes

Les avantages de la propagation locale sont l'efficacité et la facilité de compréhension du comportement des variables.

Dans un contexte d'édition, l'utilisation de plan est très intéressante lors des phases de navigation et de manipulation. En effet, lorsque l'auteur sélectionne un objet pour le déplacer le résolveur calcule un plan, c'est-à-dire qu'il calcule un sous-ensemble du graphe de contraintes qui sera suffisant pour répercuter les déformations lors du déplacement. Le résolveur oriente ce sous-graphe pour propager dans le graphe la valeur modifiée par l'auteur. Ainsi, lors de chaque déplacement élémentaire le système n'aura qu'à utiliser le plan et propager les modifications sur les autres variables affectées par la perturbation.

Les désavantages bien connus de ces méthodes sont :

- *L'incomplétude de la méthode*, c'est-à-dire que le système peut ne pas trouver de solution alors qu'il en existe une. Cela est lié par exemple à une orientation du graphe vers une valuation des variables sans solution.
- *L'impossibilité de supporter des cycles dans le graphe de contraintes*. Par conséquent de tels systèmes rejettent les systèmes de contraintes qui introduisent des cycles. Dans notre contexte, les systèmes de contraintes que l'on manipule engendrent des graphes de contraintes fortement cycliques. Des travaux ont été réalisés pour supporter des graphes cycliques. Dans Skyblue [Sannella95] par exemple, le traitement des cycles est déféré vers un résolveur spécialisé. Cependant ces techniques ne semblent pas adaptées aux graphes fortement cycliques du fait que toute la résolution du problème est déférée au résolveur spécifique, et de ce fait on ne tire plus profit de la méthode locale. Dans notre contexte, dès d'un objet appartient à un groupe d'objets, nous avons un cycle.
- *La manipulation de contraintes uniquement fonctionnelles*. Une contrainte à n variables est fonctionnelle, si lorsque l'on fixe n-1 variables, la dernière est déterminée de façon unique. Dans notre contexte, les contraintes qui limitent l'intervalle de validité d'une variable ne sont pas fonctionnelles.
- *L'absence de remise en cause du plan* calculé au début d'une opération d'édition lors d'une succession d'opérations d'édition identiques. Par exemple, tout au long du déplacement d'un objet dans la vue de présentation ce seront les mêmes déformations qui seront appliquées (Figure V-8). Si nous avons trois objets A, B et C placés les uns à côté des autres, et que nous déplaçons l'objet A vers la droite, on aimerait, dans un premier temps, retailler B, et dans un deuxième temps C, ou alors de répartir la déformation de manière équivalente sur les deux. Dans le cas d'un résolveur local, on aura B qui prendra la taille minimale autorisée par les contraintes, et après tout déplacement sera impossible. Il faudra recalculer un nouveau plan et ajouter une nouvelle contrainte pour dire que B n'est plus modifiable, et ainsi forcer le résolveur à déformer C. Pour éviter cela, il faut donc changer le plan à chaque fois que cela est nécessaire. Le résolveur devrait permettre de changer localement le plan de manière intrinsèque.

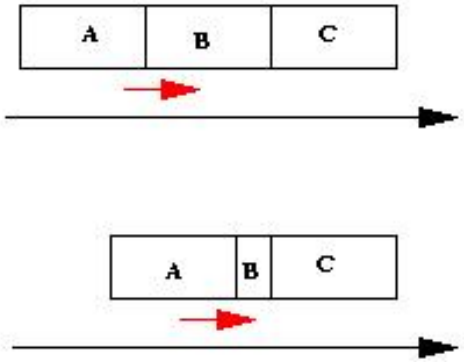


Figure V-8 : Absence de remise en cause du plan

Malgré ces limitations les techniques employées, comme la définition de poids associés aux contraintes, pour contrôler la solution et orienter la recherche sont intéressantes et adaptables à notre problème. Le résolveur Deltablue ([Sannela93]) est une implémentation de résolveur local utilisant des poids.

On peut noter que cette technique de résolution est la plus employée dans les applications graphiques. De nombreux travaux ont été faits sur le sujet et se poursuivent (ex : ThingLab [Maloney89] et SketchPad [Crilly95]).

Les résolveurs basés sur les approches locales sont intéressants du fait qu'ils ne manipulent pas à chaque étape l'ensemble du réseau de contraintes, mais seulement le sous-ensemble nécessaire.

Cependant, dans un contexte d'édition, ces méthodes sont trop limitatives pour être utilisées telles quelles, sans mettre en place un contrôle fin des résolveurs. En effet, l'impossibilité de manipuler des cycles dans le graphe de contraintes et l'absence de remise en cause du plan sont des inconvénients majeurs dans un contexte d'édition.

Néanmoins, les performances de résolution méritent qu'on étudie ces résolveurs, non pas pour les utiliser de manière globale dans l'application, mais ponctuellement pour la résolution de problèmes nécessitant de hautes performances temporelles.

6.2 Les approches globales

Nous venons de voir que les méthodes locales ne permettent pas de répondre complètement à nos besoins spécifiques. Nous étudions à présent l'autre type d'approche, les approches globales. Ces approches parcourent toutes les valeurs possibles de toutes les variables jusqu'à trouver une solution (quand elle existe.) Il existe de nombreux algorithmes [Verfaillie95] qui utilisent cette approche et qui se différencient de part leur manière de parcourir l'ensemble des valeurs. On ne présentera qu'un seul algorithme de cette classe qui est représentatif de l'approche. Cet algorithme, basé sur une technique de backtrack est utilisé dans les problèmes de satisfaction de contraintes. Il est présenté ci-dessous.

6.2.1 L'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. L'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.

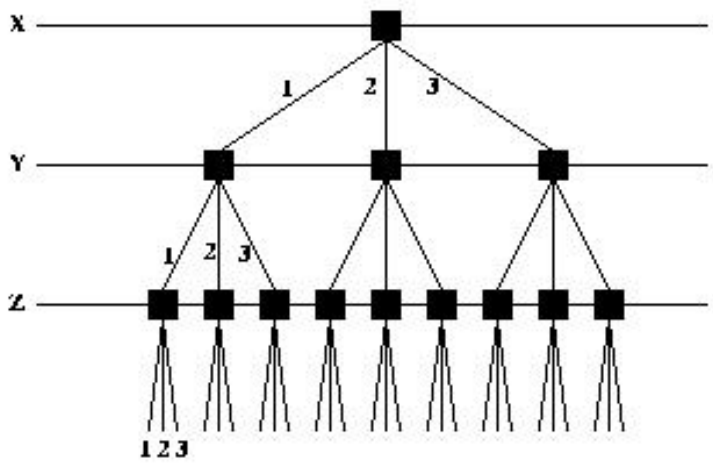


Figure V-9 : Arbre de recherche

Dans l'exemple de la Figure V-9, on peut voir un arbre de recherche possible 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.

C'est la méthode de résolution employée dans IlogSolver [IlogSolver00].

Des heuristiques sur le choix des variables à remettre en cause lors de retour arrière ou sur le choix des valeurs pour une variable sont réalisables, ce qui permet d'améliorer et de personnaliser cet algorithme de manière à orienter la recherche de solutions. Par exemple, on peut imaginer, dans notre cas, de désaffecter en priorité les variables représentant les instants de début et fin des délais.

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.

Cependant ces deux inconvénients peuvent être réduits par la remarque suivante : il est possible d'utiliser sur nos problèmes des algorithmes de filtrages pour réduire les domaines de validité de toutes les variables aux valeurs qui apparaissent dans au moins une solution du système de contraintes, et donc de diminuer l'espace de recherche améliorant ainsi les coûts d'un algorithme de type backtrack. Un autre avantage de l'utilisation d'un algorithme de filtrage, est que celui-ci permet de connaître l'intervalle de validité des variables.

Il existe de nombreux travaux pour améliorer ce mécanisme de résolution en intégrant par exemple la connaissance d'une résolution précédente pour améliorer la résolution courante. Ces mécanismes sont très intéressants dans un contexte d'édition incrémentale. Ces travaux sont résumés et les différents algorithmes présentés dans [Verfaillie95].

6.2.2 jSolver : un implémentation basée sur le backtrack

Nous allons illustrer ces approches par la présentation du résolveur jSolver [HonWai99]. jSolver est une librairie Java qui permet de manipuler et de résoudre des CSP. Chaque variable est définie dans un domaine entier fini. L'algorithme de recherche se décompose en quatre étapes (Figure V-10):

1. Choisir une variable dans l'ensemble des variables à évaluer.
2. Choisir une valeur pour la variable dans le domaine, s'il n'y a plus de valeur possible un retour arrière intervient.
3. Affectation de la valeur à la variable.
4. Propagation des contraintes.

L'implémentation de jSolver nous a permis de définir différentes stratégies pour les étapes 1 et 2. En effet, jSolver permet d'ordonner les variables à évaluer ainsi que les domaines de définition des variables. On peut ainsi prendre en compte plus facilement des préférences de l'auteur vis-à-vis des déformations (minimisation du nombre d'objets déformés, localité des déformations,).

Outre ces fonctionnalités très intéressantes dans notre contexte d'édition, jSolver offre un mécanisme de planification à la Deltablue (section 6.1). Ce mécanisme, bien contrôlé, peut être très utile pour certaines opérations d'édition répétitives. On peut noter que par rapport à Deltablue, jSolver est complet. C'est-à-dire que s'il existe une solution, il la trouve.

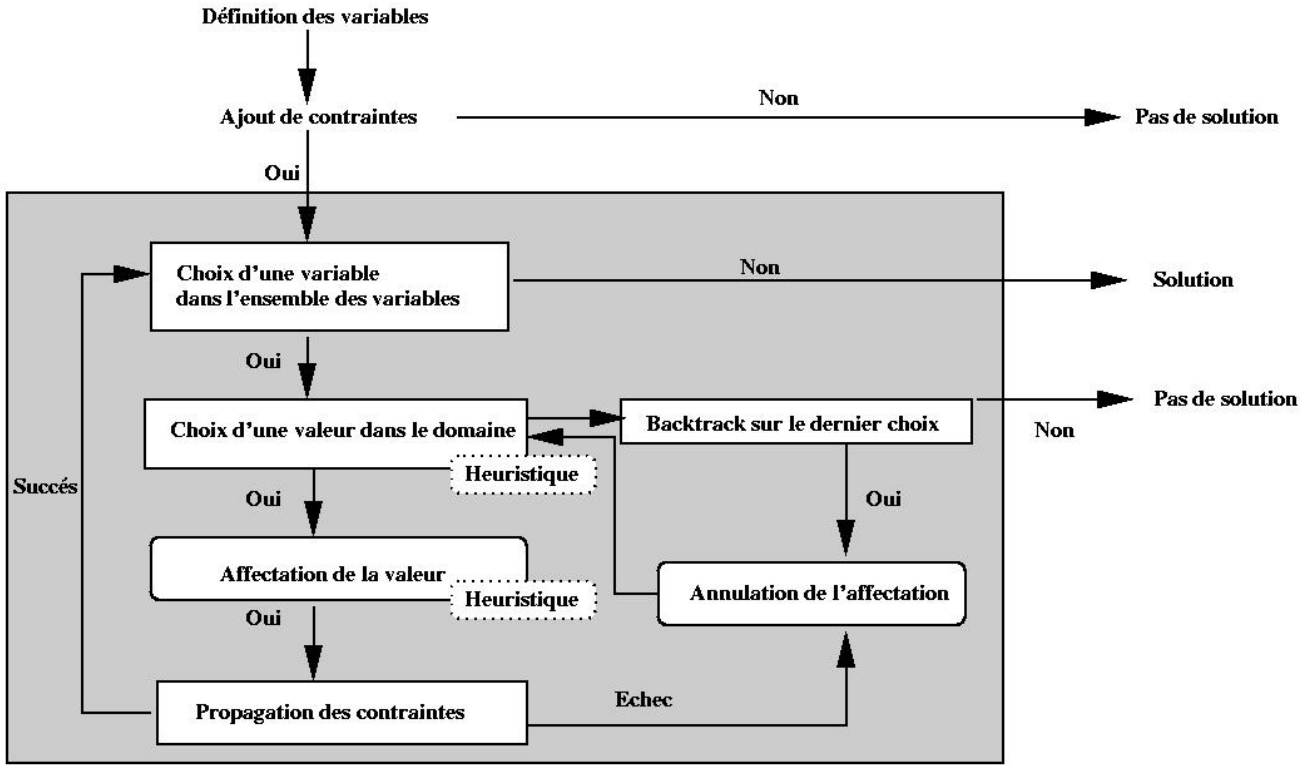


Figure V-10 : jSolver

6.3 Les approches basées sur l'algorithme du simplex

De nombreux travaux ont permis de résoudre des problèmes linéaires. Le premier algorithme permettant de les résoudre est le simplex réalisé par Dantzig dans les années 40 [Dantzig49]. 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)*. La résolution de l'algorithme devra maximiser la valeur de cette fonction tout en satisfaisant l'ensemble de contraintes. Les algorithmes que nous allons présenter par la suite sont basés sur le Dual-simplex [Dantzig54, Marriott98].

6.3.1 Cassowary

Cassowary est un résolveur de contraintes qui traite des équations et des inéquations linéaires. Cet algorithme a été développé à l'université de Washington par Badros et Borning ([Badros98]).

L'algorithme se décompose en deux phases :

1. *Phase de prétraitement :* cette phase vise à traduire le problème sous une forme ne contenant ni inégalité, ni variable négative, de manière à résoudre l'ensemble de contraintes obtenu par le simplex.

- Les inégalités de la forme $x \geq y$ sont traduites sous la forme $x = a + y$.
- Les variables négatives (*v*), sont substituées par des équations de la forme *e* = -*v*, on substitue ensuite toutes les occurrences de *v* par -*e*. La résolution des variables *v*, se fera dans une étape postérieure, une fois la valeur de *e* connue.

1. *Optimisation pour le simplex* : les contraintes sont mises sous forme de matrice. Une optimisation de la matrice est réalisée dans le but de favoriser la recherche d'une solution optimale et de limiter le nombre de variables manipulées par le simplex. Cette optimisation se base en partie sur les poids associés aux contraintes. Ce mécanisme complexe est présenté complètement dans [Badros98].
2. *Une solution* est ensuite obtenue par la méthode du simplex.

Les travaux réalisés ont aussi eu pour but de rendre incrémentales les opérations d'ajout et de retrait de contraintes. Ces travaux sont rendus complexes par l'optimisation des données pour le simplex. En effet, il est nécessaire lors de l'ajout et du retrait d'équation de maintenir une cohérence entre le problème initial est sa version optimisée.

6.3.2 QOCA

QOCA [Marriott98b] est une boîte à outils pour résoudre les CSP qui est construite au-dessus de Cassowary. QOCA utilise Cassowary pour résoudre les contraintes et introduit des notions supplémentaires pour contrôler la solution trouvée par le résolveur.

Cette boîte à outils est basée sur un modèle de normalisation de l'espace de solution. Cette normalisation, permet de :

- définir une mesure qui donne la distance entre deux affectations d'une même variable (dans des solutions différentes). Cette mesure est proche du paramètre P_2 de la fonction de distance que nous avons défini (section 5.3.1).
- définir une mesure de distance entre deux solutions de l'ensemble de contraintes, à partir de la mesure de la distance entre deux affectations de variables.

La difficulté consiste à maintenir la normalisation de l'espace lors d'opérations modifiant le système de contraintes, ces opérations sont :

- Ajout de contraintes au jeu de contraintes courant, dans ce cas, l'affectation des variables est celle qui est la plus proche possible de la solution courante. La notion de plus proche est définie grâce à la métrique de l'espace.
- Retrait d'une contrainte, dans ce cas, la solution courante reste inchangée.
- Modification de ma solution courante, dans ce cas l'auteur peut suggérer un ensemble de valeurs pour les variables, et le système choisit la nouvelle solution de manière à ce que celle-ci soit la plus proche possible de celle suggérée.

La boîte à outils a été utilisée avec des outils comme Idraw [Helm95], et donne des résultats très satisfaisants. Cependant, dans notre contexte, cette approche présente quelques limites. Par exemple, le fait de ne pas réévaluer le système de contraintes lors du retrait d'une contrainte pose des problèmes lors de l'utilisation des domaines de validité. En effet dans notre contexte, lors du retrait d'une contrainte, l'intervalle de validité des variables doit être recalculé.

6.4 Résolveurs spécifiques développés dans Kaomi

Pour tirer profit de la structure interne de Kaomi qui comporte un graphe temporel (voir chapitre IV), deux algorithmes spécifiques ont été implémentés dans le projet. Le premier est une adaptation de PC2 (section 6.4.1) pour vérifier la cohérence temporelle et formater une hiérarchie de graphes (réalisé par F. Bes [Bes98]). Le second, que j'ai réalisé, est un algorithme permettant à l'auteur d'effectuer des modifications dans la vue temporelle, pour déplacer et retailler les objets dans cette vue ([Tardif97]).

6.4.1 PC2 adapté

L'ensemble des contraintes temporelles peut être directement traduit dans un graphe temporel orienté et acyclique si l'on reste dans le cadre des STP (Simple Temporal Problem). La notion de STP, qui est une restriction des CSP a été introduite par Dechter ([Dechter91]). Un STP est un ensemble de contraintes où toutes les variables représentent des instants temporels et toutes les contraintes s'expriment sous la forme de différences bornées. De ce fait, il ne peut supporter les relations de causalité. Dans le chapitre IV (section 6.3) nous avons présenté la construction du graphe correspondant. Les noeuds sont les instants de début et de fin des intervalles. La présence d'un arc entre deux noeuds n_1 et n_2, traduit le fait que n_2 se situe temporellement après n_1. Les arcs sont étiquetés par trois valeurs :

- La durée prévue de l'objet.
- La borne inférieure de l'intervalle de flexibilité de l'objet dans le scénario.
- La borne supérieure de l'intervalle de flexibilité de l'objet dans le scénario.

Dans l'exemple de la Figure V-11, on a trois objets A, B, C. Ces trois objets ont des durées respectives de : A[1,2,4], B[3,5,9], C[2,3,5]. Il y a trois relations entre ces objets : *Starts* (A, B), *Meets*(A, C), *Finishes*(C, B).

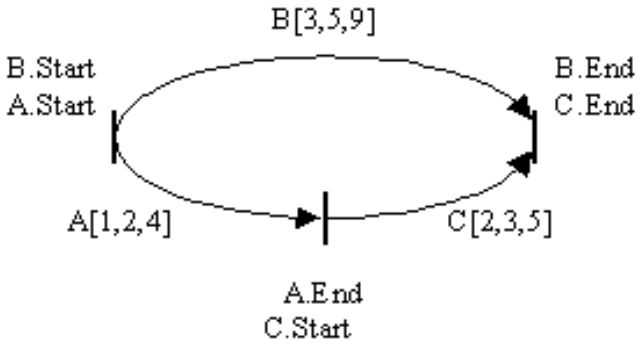


Figure V-11: Un graphe temporel

Pour vérifier la cohérence de graphes de contraintes, de nombreux algorithmes ont été proposés. On peut citer AC3 ([Mackworth77]), AC4 ([Mohr86]), DPC ([Dechter88]) ou AC6 ([Bessière94]). Ces algorithmes sont basés sur la notion de cohérence d'arcs ou de cohérence de chemins. PC2 [Mackworth77] est un algorithme basé sur la cohérence de chemins qui filtre les intervalles de durée associés aux arcs de manière à supprimer les valeurs qui n'appartiennent à aucune solution.

PC2 adapté au graphe temporel

Une adaptation de PC2 a été utilisée par Dechter ([Dechter91], [Dechter88]) pour vérifier la cohérence de graphes temporels.

PC2 se base sur un graphe temporel complet, et considère tous les chemins de longueur inférieure ou égale à deux entre deux noeuds, pour filtrer les valeurs qui n'appartiennent à aucune solution. Un état incohérent est détecté quand toutes les valeurs d'un intervalle ont été éliminées.

L'algorithme peut être décrit comme suit :

1. Complétion du graphe, jusqu'à obtenir un graphe complet. Cette étape permet de rendre possible le calcul de tous les chemins de longueur inférieure ou

égale à deux entre 2 noeuds. Les arcs introduits ont une durée initiale de [-∞ , +∞].

2. Pour chaque couple de noeuds :

- La durée de chaque chemin de longueur inférieure ou égale à deux est calculée.
- L'intersection de ces durées est comparée avec la durée de l'arc reliant ces deux noeuds.
- Si cette intersection est vide, une incohérence est détectée, sinon la valeur de l'intersection est prise comme nouvelle durée de l'arc entre les deux noeuds.

L'étape 2 est répétée jusqu'à l'obtention d'une stabilité des durées des arcs du graphe.

Le résultat de cet algorithme est un graphe minimal pour le jeu de contraintes donné. Les intervalles sur les arcs représentent les intervalles de validité des variables de durée. C'est-à-dire que si l'on choisit une valeur dans un des intervalles, alors cette valeur appartient à une solution. On peut donc trouver une valuation pour toutes les durées des autres arcs.

Un formateur basé sur PC2 [Bes98, Layaïda97]

Cette propriété nous permet d'utiliser PC2 comme formateur, en faisant une succession de filtrages et d'affectations.

Dans une approche telle que PC2 l'ajout de contraintes est relativement simple grâce à la construction du graphe, cependant, le retrait d'une contrainte est beaucoup plus difficile du fait que les domaines ont été filtrés. Il faut donc réintroduire ces domaines. L'implémentation de PC2 dont nous disposons à l'heure actuelle oblige, lors de chaque retrait de contrainte, à réintroduire les domaines non filtrés et à filtrer le graphe. On peut noter que des méthodes pour permettre de réduire les réintroductions de valeurs dans les domaines ont été étudiées et évaluées principalement sur AC3 et AC4 ([Bessière91]) et pourraient être implémentées dans PC2.

Adaptation de PC2 pour une hiérarchie de graphes (HPC2) [Bes98]

Avec la définition d'une hiérarchie d'objets temporels, la dimension temporelle d'un document ne se représente pas par un graphe mais par une hiérarchie de graphes. Un arc dans un graphe à un niveau donné représente un objet ou un graphe du niveau inférieur. La propagation d'une contrainte se fait dans un premier temps au niveau du graphe dans laquelle elle a été introduite, et ensuite elle est propagée dans les niveaux supérieurs et inférieurs jusqu'à obtenir une stabilité de la hiérarchie de graphe.

Dans la Figure V-12 je présente l'algorithme HPC2 ([Bes98]) proposé pour manipuler de telles hiérarchies.

Cet algorithme tire profit des domaines minimaux calculés par PC2 pour réduire la propagation entre les graphes.

Nous allons présenter deux versions qui sont utilisées dans Kaomi :

- La première incrémentale, qui permet de vérifier la cohérence à chaque insertion de relation en tirant profit du formatage à l'étape précédente. Dans ce cas, il faut vérifier la cohérence au niveau où la contrainte a été insérée (ligne 10), il faut ensuite faire une propagation dans tous les sous-graphes (ligne11), et ensuite remonter cette valeur sur le composite englobant (ligne 12). Cette version de l'algorithme est utilisée pour les opérations d'édition (ajout) de l'auteur. Elle n'est pas utilisée pour le retrait d'objet à cause des limitations de la version courante de PC2.
- La deuxième non incrémentale, permet de vérifier la cohérence d'un graphe quelconque. Dans ce cas, il faut vérifier la cohérence d'abord dans les arbres les plus bas dans la hiérarchie (ligne 5), et ensuite faire propager les valeurs remontées entre frères (ligne 7). Cette version de l'algorithme est utilisée après la phase de lecture du fichier source. En effet, dans le contexte d'ouverture du document, nous appelons le mécanisme de résolution à la fin de la construction du graphe, et de ce fait, cela a nécessité une adaptation de l'algorithme à cette utilisation spécifique.

```
1.  HPC2_incremental(Graphe G) {
2.  Vérifier_cohérence(G);
3.  }
4.  HPC2_global(Graphe G) {
5.  Reduire_BottomUpFirst(G);
6.  Pour chaque G1 ÎW (G)
7.  Reduire_TopDown(G1);
8.  }
9.  Vérifier_Cohérence (Graphe G)
10. PC2(G);
11. Pour chaque G1 ÎW (G)
12. Reduire_TopDown(G1)
13. Vérifier_Cohérence(Parent(G))
14. }
```

```
● Reduire_BottomUpFirst(Graphe G)
● Pour chaque G1 Î W (G)
● ReduceBottomUpFirst(G1)
● PC2(G1)
● }
● Reduire_TopDown(Graphe G) {
● PC2(G)
● Pour chaque G1 Î W (G)
● Reduire_TopDown(G1)
● }

W (G)= {Graphes représentés par un arc dans G
pour lesquels la durée a été modifiée}
```

Figure V-12 : Algorithmes HPC2

6.4.2 Navigation dans la vue temporelle : l'algorithme AVT

La vue temporelle de Kaomi, comme nous l'avons présentée dans le chapitre précédent, offre une visualisation du scénario temporel du document, elle permet donc d'afficher une solution du réseau de contraintes. De plus, cette vue permet à l'auteur de naviguer dans l'espace de solutions. Lors de mon projet de DEA, j'ai défini un algorithme qui prend en charge cette tâche de navigation. Nous l'appellerons par la suite AVT (Algorithme de la Vue Temporelle).

L'objectif de l'AVT est de calculer le plus rapidement possible une nouvelle solution à chaque action de l'auteur dans la vue temporelle. Cet algorithme est donc complètement spécialisé pour arriver à cet objectif.

L'algorithme réalisé nécessite la connaissance des intervalles minimaux sur les arcs du graphe. Il est donc nécessaire avant d'appliquer l'algorithme de la vue temporelle qu'un autre algorithme du type PC2 calcule les domaines minimaux. Avec cette hypothèse, on peut assurer que si l'auteur choisit une des valeurs d'un des domaines minimaux, alors il existe une solution au système de contraintes.

L'AVT est basé sur le graphe temporel résultant de la spécification de l'auteur et non pas sur le graphe de contraintes. Cet algorithme est présenté complètement dans [Tardif97], je ne rappelle ici que son principe. La méthode utilisée est une propagation de la modification vers les intervalles les plus proches capables

d'absorber cette modification. C'est un algorithme en deux étapes :

1. Collecte d'information : en fonction du jeu de contraintes et de l'action de l'auteur (telle que déplacement ou retaillage d'un objet), le résolveur calcule l'intervalle maximal pour le déplacement (ou le retaillage) de cet objet. Il calcule aussi les objets qui seront affectés (déplacés ou retaillés) par cette modification.
2. Propagation de l'action de l'auteur : chaque fois que l'auteur fait une action élémentaire (par exemple à chaque étape d'un déplacement), le système utilise les informations calculées à l'étape précédente pour appliquer les déformations. On peut comparer cette approche avec l'utilisation des plans dans Deltablue.

6.5 Bilan des approches

L'étude bibliographique de cette section des différentes méthodes de résolution nous amène a des conclusions générales sur l'utilisation des résolveurs de contraintes.

Les algorithmes globaux présentés semblent pouvoir répondre complètement à nos besoins d'expressivité. Cependant les performances temporelles de tels algorithmes restreignent leur utilisation.

Les algorithmes locaux qui ont la réputation d'être très performants sont par contre incomplets et ne peuvent être utilisés que partiellement pour répondre à notre problème.

Enfin, les algorithmes à base de simplex semblent offrir le meilleur compromis entre expressivité et rapidité.

Cette première analyse doit être et sera complétée par une analyse quantitative et qualitative dans un contexte d'édition (section 8).

Nous allons maintenant dans un premier temps, expliquer comment s'inscrivent les résolveurs de contraintes dans la boîte à outils Kaomi (section 7), nous présenterons ensuite l'évaluation des différents résolveurs dans leur contexte d'utilisation (section 8), et nous conclurons ce chapitre en faisant un bilan des différentes utilisations possibles d'une technologie à base de contraintes dans un contexte d'édition de documents multimédias.

7 Le module contraintes de Kaomi

Le module de contraintes de Kaomi a été implémenté comme un service de la boîte à outils et n'a donc de ce fait pas été intégré directement dans chacun des modules utilisant un résolveur de contraintes (vue temporelle, vue d'exécution, module d'édition) (Chapitre IV section 5.2).

Cela a été réalisé pour deux raisons :

- Permettre une grande souplesse dans le remplacement des résolveurs et l'intégration de nouveaux résolveurs. Cela nous a permis de tester facilement les différents résolveurs dans les différents contextes d'utilisation.
- Changer de résolveurs en fonction du type de manipulation ou de besoins que l'on a. C'est-à-dire que nous disposons d'un module, qui à partir de certaines informations (type d'utilisation, type de données) nous permet d'appeler un résolveur, sans pour autant savoir lequel on utilise réellement à un instant t. Cela permet, par exemple, d'utiliser un résolveur différent lors de l'ajout d'une relation et lors d'une manipulation dans la vue temporelle où les besoins en termes de performances temporelles sont plus importants.

Au cours de cette section nous présenterons dans un premier temps l'architecture du module de contraintes (section 7.1) et plus particulièrement l'organisation

hiérarchique des résolveurs de contraintes (section 7.2). Dans un deuxième temps nous présenterons les politiques de formatage choisies dans Kaomi (section 7.3).

Enfin dans la section 7.4 nous présenterons comment se réalise l'intégration d'un résolveur de contraintes dans Kaomi.

7.1 Architecture du module de contraintes

La structure du module de gestion de contraintes est la suivante dans Kaomi :

- Un ensemble de résolveurs avec des propriétés et une structure de données qui leur sont propres.
- Un module de choix qui permet de fournir un résolveur lorsqu'un module désire en utiliser un.

Ce module de contraintes peut être utilisé par toute l'application. Par exemple, à l'heure actuelle, quatre modules l'utilisent : le module de présentation spatial dans la vue d'exécution, les modules temporels dans les vues d'exécution et temporelle, ainsi que le module édition de Kaomi.

Le processus général d'utilisation du module contraintes est décrit dans la Figure V-13. Ce processus est le suivant.

Lorsque l'auteur effectue une modification sur son document, par exemple, une action d'édition (transition 1), cette opération d'édition est appliquée dans le module d'édition. Ce module met à jour sa structure de données (transition 2). De manière à garantir la cohérence du document et à propager la modification de l'auteur, on fait appel au résolveur de contraintes pour calculer l'ensemble des modifications à faire (transition 3). Le résolveur calcule alors cet ensemble de modifications et les propage à la structure de données du document (transition 4). Une fois la structure de données à jour, celle-ci indique au module d'édition que le calcul est terminé (transition 5), et ce dernier peut demander la mise à jour de l'affichage (transition 6).

Dans le cas d'une modification locale à une vue, le processus est similaire. Les modifications se font localement sur les structures de données de la vue. Le processus met alors en oeuvre les transitions 7 à 12.

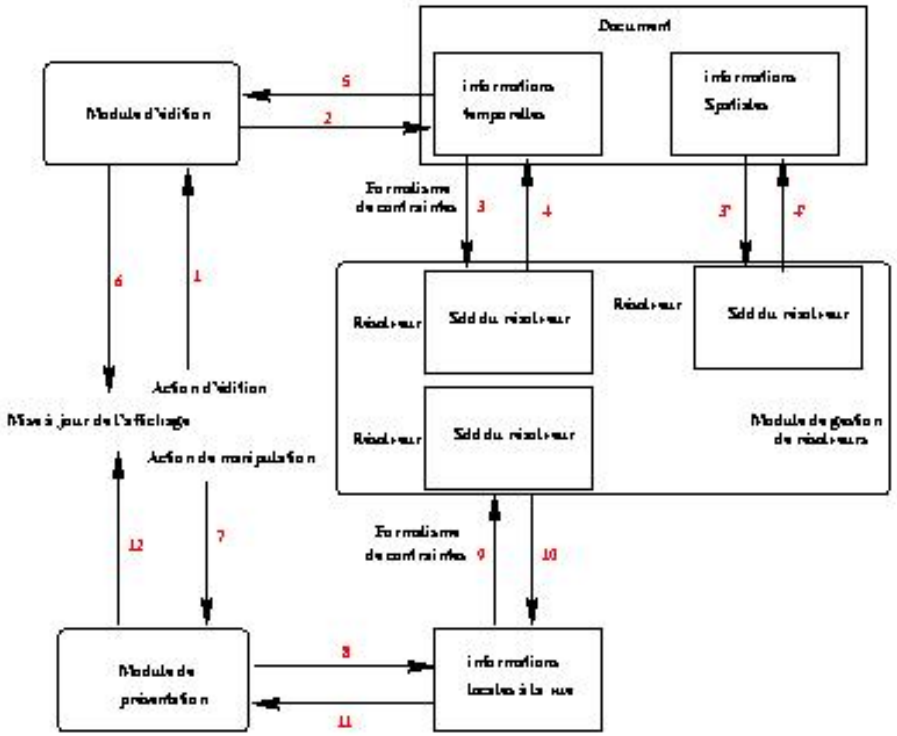


Figure V-13 : Maintiens des relations lors de l'édition

7.2 Organisation hiérarchique des résolveurs

Dans le cadre d'une décomposition hiérarchique du document, on utilise un résolveur par niveau de hiérarchie, et on utilise un algorithme qui utilise le même principe de propagation hiérarchique que celui présenté dans HPC2 (voir chapitre V) pour propager les valeurs entre les différents niveaux de hiérarchie. Dans l'exemple de la Figure V-14, on peut voir une hiérarchie temporelle avec les différentes instances de résolveurs.

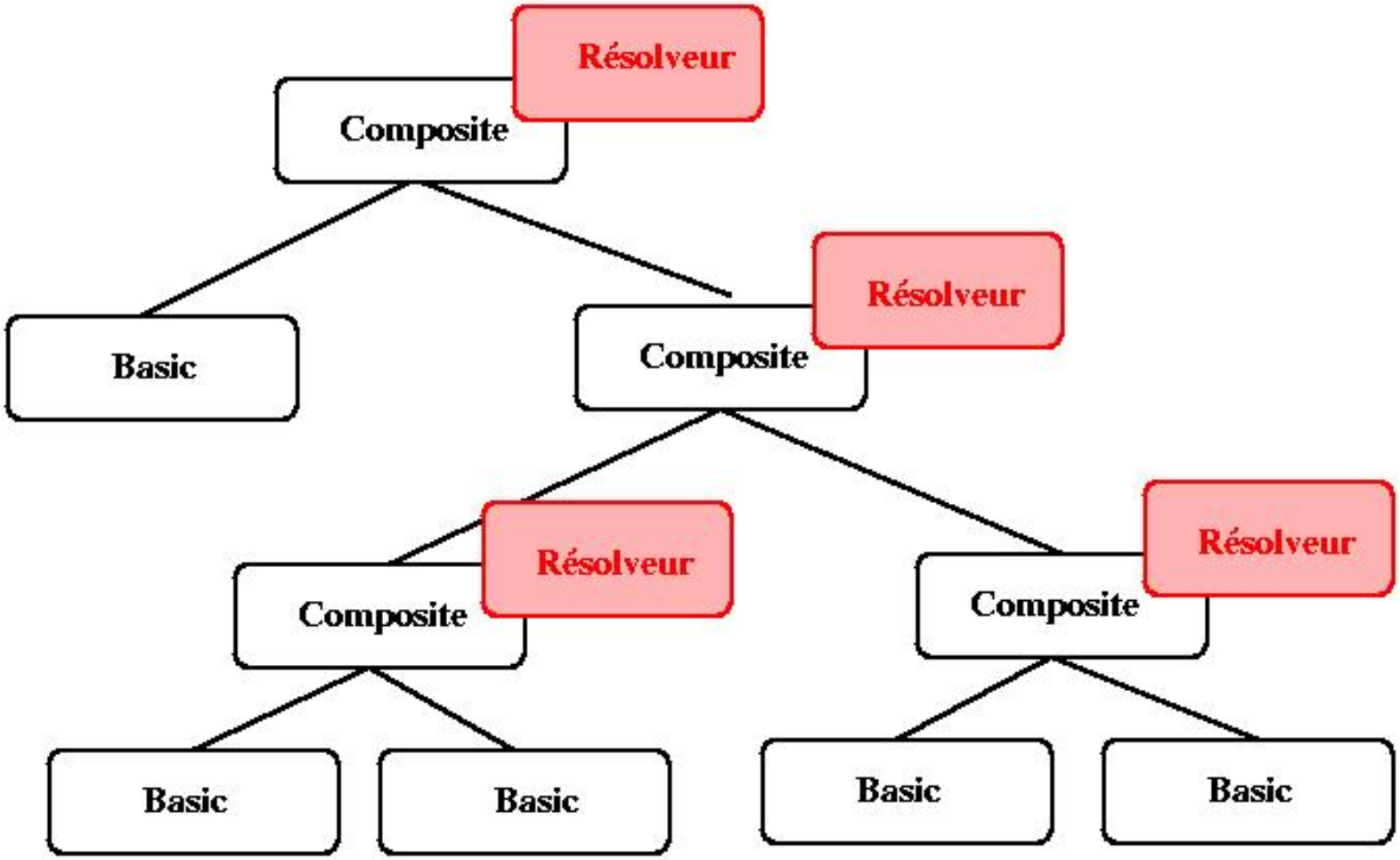


Figure V-14 : Hiérarchie de résolveurs

Nous rappelons qu'à chaque étape, l'hypothèse est : les domaines de validités sur les intervalles de valeurs associés aux objets sont maintenus.

Lors d'une modification de valeur le résolveur hiérarchique s'occupe de limiter les propagations seulement aux résolveurs pour lesquels c'est nécessaire. Par exemple, dans la Figure V-15, nous illustrons deux situations :

- Dans le premier cas, l'auteur a modifié une valeur sur un objet de base, cette modification n'a pas remis en cause les domaines minimaux du composite, de ce fait, cette opération n'a nécessité qu'un seul formatage.
- Dans le deuxième cas, l'auteur modifie la valeur d'un des attributs d'un objet de base, le résolveur du composite supérieur calcule une nouvelle solution. Si ce calcul modifie l'intervalle minimal du composite, l'information est propagée vers le composite supérieur. Dans le cas où l'intervalle minimal n'est pas modifié, l'information n'est pas propagée vers le composite ascendant. Dans les deux cas, le nouveau formatage est propagé à tous les descendants. Si un

des descendants est un composite, alors cela nécessite une phase de formatage. Cette phase de formatage, du fait du maintien des domaines minimaux ne nécessitera pas une remise en cause des formatages de niveaux supérieurs déjà effectués.

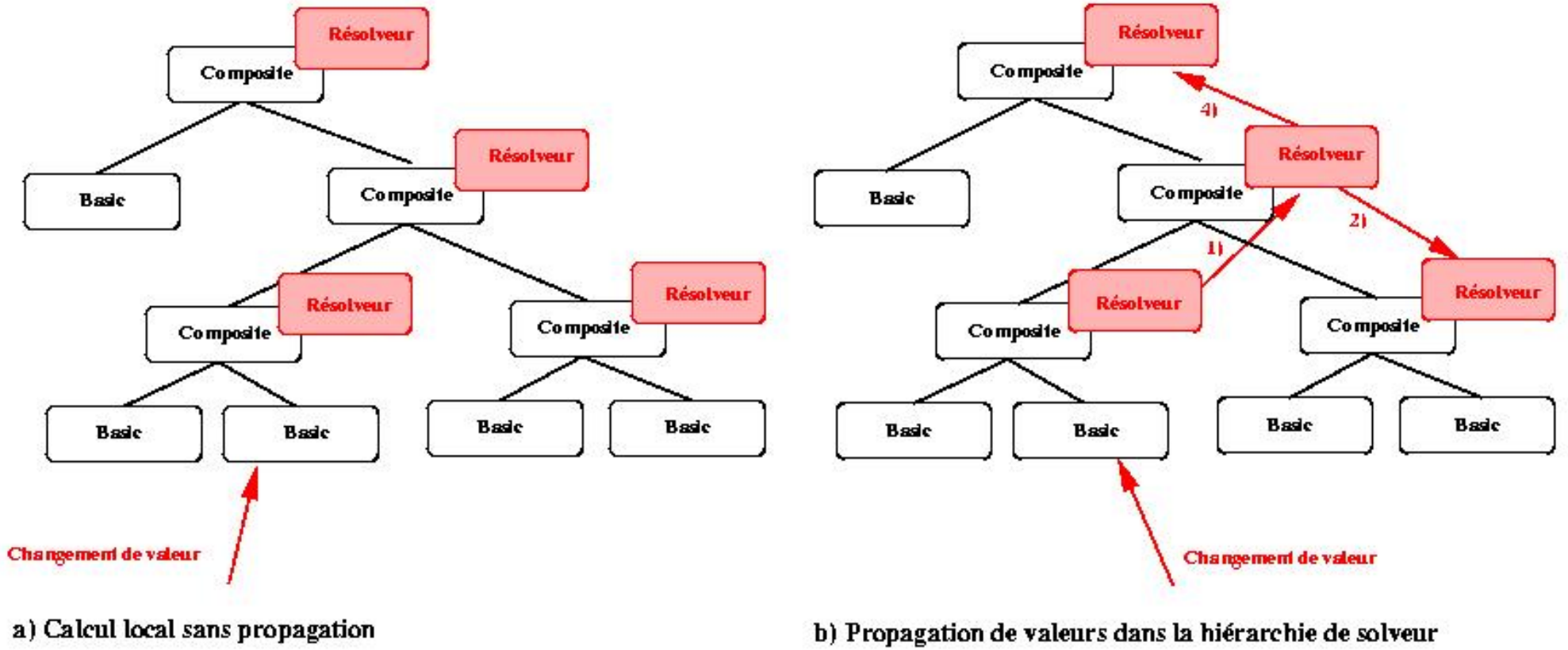


Figure V-15 : Propagation des valeurs

On peut voir qu'un des avantages de cette structuration est de limiter les calculs des résolveurs de contraintes.

On peut noter que des langages comme SMIL2 ou MHML permettent de définir des événements entre tous les objets d'un document, et plus particulièrement entre deux objets qui ne sont pas initialement dans le même composite temporel. Le système doit alors recalculer une nouvelle décomposition hiérarchique en fonction des relations contenues dans le document, telles que, si deux objets ont une relation, ou un événement entre eux, ils sont dans le même composite. Dans des cas extrêmes, on obtient à la fin de ce processus un seul composite avec tous les objets du document à l'intérieur.

7.3 Politique de formatage : compromis dans l'utilisation des résolveurs de contraintes

De par les différentes expériences d'utilisation de résolveurs de contraintes qui ont été faites dans le cadre de l'édition de documents multimédias ([Carcone97b], [Badros98]), nous savons que tous les besoins ne peuvent être satisfaits par le même résolveur. On doit donc définir un certain nombre de compromis. Nous allons décrire un ensemble qui va nous servir à restreindre notre problème, et les expliquer en fonction des objectifs que l'on souhaite atteindre.

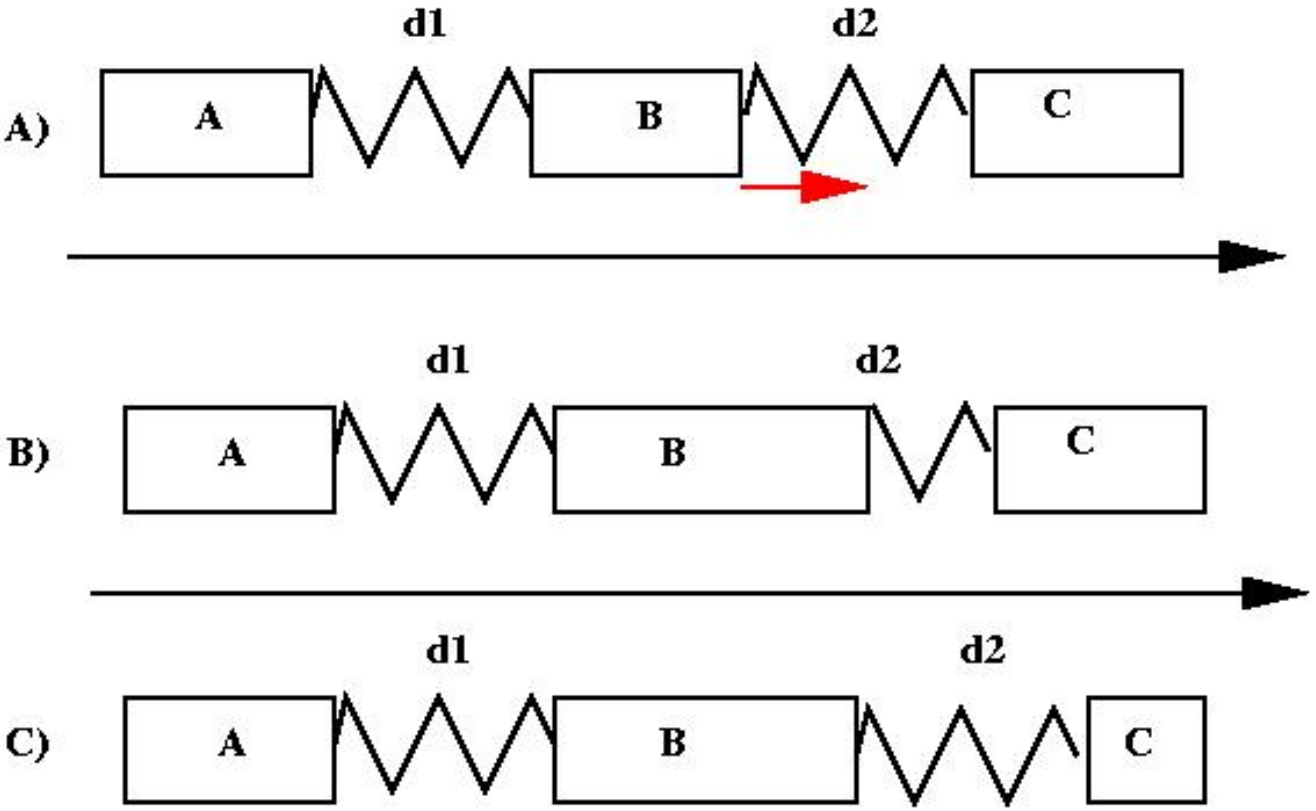
7.3.1 Choix entre performances en temps de calcul et pertinence de la solution

Le besoin essentiel, dans un environnement auteur, est de trouver rapidement la meilleure solution du réseau de contraintes. On sait que ce problème est complexe. De manière à réduire les temps de calcul, on peut manipuler une définition relâchée de la meilleure solution, et se contenter de chercher une *bonne* solution qui ne sera pas forcément la meilleure.

Cette solution peut être calculée en utilisant des fonctions heuristiques dans les approches globales (jSolver), ou en orientant l'évaluation dans les techniques locales (Deltablue).

Par exemple, dans le cas où l'on a trois objets A, B, C, avec les relations A `Before` B et B `before` C (Figure V-16A). Si l'auteur décide de retailler l'objet B par la droite. Le système a plusieurs manières de calculer une nouvelle solution, il peut par exemple :

- réduire le délai d2 (Figure V-16B) ;
- réduire la durée de l'objet C si celui-ci est flexible (Figure V-16C) ;
- réduire le délai d1 (Figure V-16D) ;
- réduire l'objet A si celui-ci est flexible (Figure V-16E).



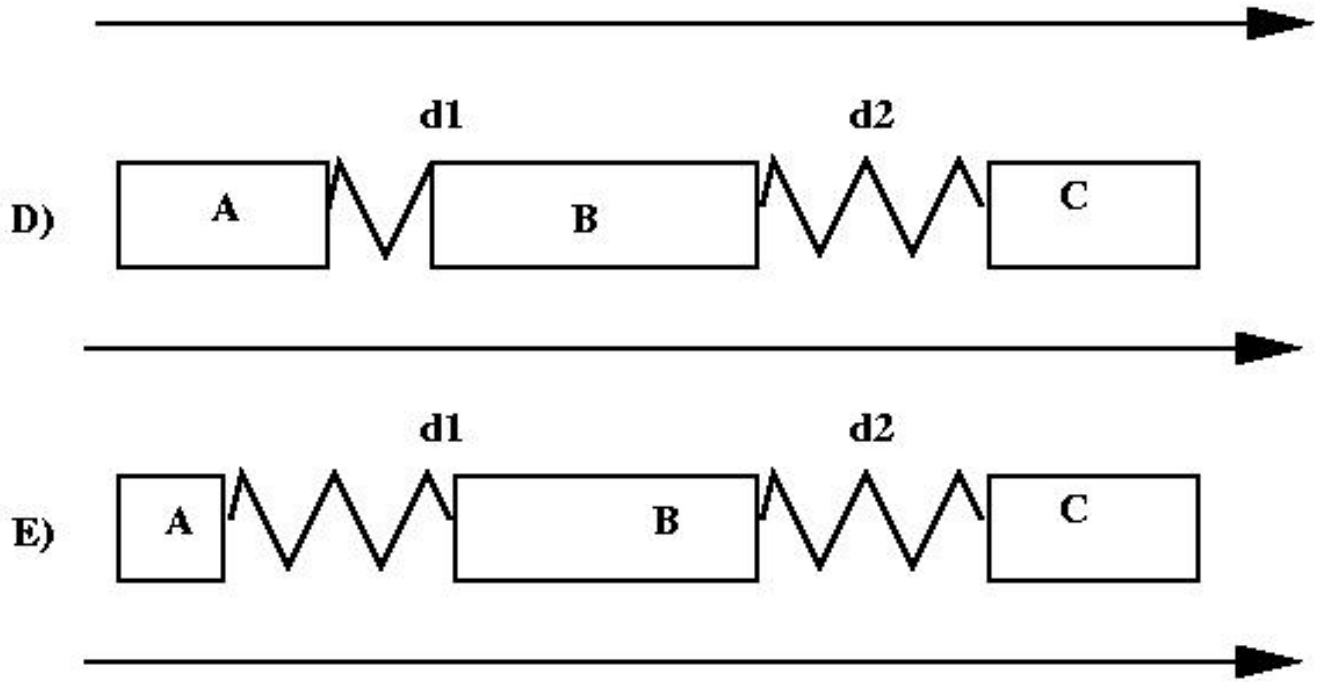


Figure V-16 : Solutions multiples

La meilleure solution dépend en partie de la manière dont l'auteur a spécifié ses relations. Par exemple, s'il a explicitement spécifié les durées d1 et d2, alors une bonne solution ne modifiera pas ces deux durées.

Si dans un contexte donné, la meilleure solution est la solution B, alors, le fait de ne pas insérer de contraintes pour orienter la recherche de solution vers cette solution a pour effet de diminuer sensiblement le nombre de contraintes introduites dans le résolveur. La recherche de solution sera d'autant plus rapide. Cependant, on risquera de tomber sur des solutions moins bonnes. Toute la difficulté consiste donc à trouver un bon compromis.

7.3.2 Choix entre temps de calcul et possibilités d'édition par des manipulations directes

Une des manières d'obtenir de bonnes performances temporelles, lors de la résolution d'un système de contraintes, est d'introduire des restrictions dans les opérations que l'on permet à l'auteur de réaliser. Par exemple, dans la dimension spatiale, un des gains les plus significatifs en temps est obtenu en interdisant à l'auteur de modifier la taille d'un objet composite, par manipulation d'un de ses fils. De ce fait, les modifications peuvent se faire localement, et ne remettent pas en cause la formatage de tout le document.

Si l'auteur désire modifier la taille d'un objet composite, il peut toujours la modifier en sélectionnant directement l'objet composite ou en modifiant un de ces frères dans la structure spatiale.

Le petit exemple est présenté dans la Figure V-17 pour illustrer une telle limitation : l'auteur ne peut déplacer l'objet B verticalement du fait qu'il définit la boîte englobante minimale de l'objet composite. Cependant, l'auteur peut le déplacer horizontalement. Pour la même raison, il peut déplacer l'objet C verticalement

mais pas horizontalement. L'objet D, lui, ne peut pas bouger car il définit deux des côtés de la boîte englobante minimale. Dans cet exemple, la boîte englobante minimale est définie de manière dynamique par les objets qu'elle contient.

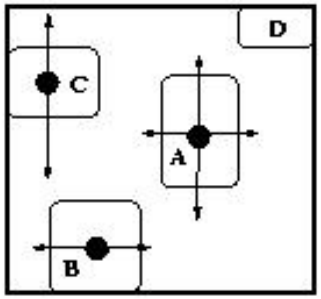


Figure V-17: Opérations d'édition spatiale sur des objets à l'intérieur d'un composite

L'auteur peut aussi utiliser des opérations d'édition indirectes pour, par exemple, retailler un objet composite, comme changer ses attributs Hauteur et Largeur dans la vue attribut. L'auteur peut ainsi changer la taille des objets composite ; on ne limite donc pas l'espace de solutions possibles de son document.

Maintenir une boîte englobante nécessite des résolveurs capables de traiter les cycles dans un graphe de contraintes. Maintenir la boîte englobante minimale n'est pas une chose facile avec les résolveurs linéaires, car cela ne s'exprime pas de manière linéaire. Il existe cependant plusieurs manières de contourner ce problème. La première solution consiste à manipuler une définition plus souple de la boîte englobante d'un composite : c'est-à-dire que l'on considère une boîte englobante qui n'est pas forcément la boîte minimale. Comme dit précédemment la seconde manière est d'interdire le retaillage des composites.

7.3.3 Choix entre temps de calcul et pouvoir d'expression

On peut améliorer les performances temporelles de la gestion de contraintes en réduisant le jeu de relations que peut définir l'auteur. Par exemple, certains résolveurs locaux sont très performants, mais ne sont pas capables de maintenir des réseaux de contraintes cycliques. De ce fait si on désire avoir de hautes performances, il faut supprimer les relations telles que "centrer" ou les relations hiérarchiques qui introduisent des cycles dans le système de contraintes.

7.3.4 Choix faits dans Kaomi

Nous avons fait dans Kaomi un choix à deux niveaux :

- Du côté du document, où nous sommes obligés d'être complet si l'on ne veut pas trop restreindre le pouvoir d'expression, nous utilisons des résolveurs complets.
- Dans la vue temporelle et dans la vue spatiale, nous limitons l'auteur dans les déformations autorisées par manipulations directes. On interdit, par exemple, le retaillage implicite de composites qui est une opération coûteuse en temps. De plus, dans la vue temporelle, par exemple, les déformations sont limitées au voisinage proche de l'objet manipulé.

7.4 Intégration d'un résolveur de contraintes dans Kaomi

Pour l'instant, chaque module (module d'édition, vue temporelle, vue d'exécution, vue spatiale) n'utilise qu'un résolveur de contraintes. Si nous voulions utiliser plusieurs résolveurs dans le même module, de manière à exploiter au mieux les capacités des résolveurs en fonction du contexte et de l'opération d'édition réalisée, cela demanderait un maintien de la cohérence des informations partagées entre les différents résolveurs.

7.4.1 Association module / résolveur

Comme nous avons pu le voir dans le chapitre précédent, chacun des résolveurs de contraintes a des capacités et des performances qui dépendent énormément du contexte d'utilisation. Nous verrons dans la section 8 une évaluation qualitative et quantitative de ces performances.

Dans le cadre de Kaomi, nous avons la possibilité, lors de la création d'un résolveur par un module, de choisir le résolveur en fonction de certains critères. Par exemple, le module VueTemporelle, lors de son initialisation, va demander au module de gestion de résolveurs un résolveur temporel. Le module VueTemporelle peut préciser sa demande en indiquant s'il veut un résolveur optimal pour l'édition ou pour la manipulation. En fonction des différentes informations, le module de gestion de résolveurs fournira à la vue temporelle le résolveur le plus adapté à ces besoins. Dans Kaomi, nous utilisons en permanence plusieurs types de résolveurs (locaux, globaux, spécialisés pour une application spécifique), chacun étant utilisé de manière à avoir des capacités optimales.

7.4.2 Vues temporelle et vue d'exécution

La vue d'exécution utilise directement le résultat du formatage sur le document pour placer les objets spatialement et pour les exécuter temporellement. Lorsque l'auteur déplace un objet dans la vue spatiale, cela est considéré comme une opération d'édition, et on fait appel au résolveur associé au document pour calculer une nouvelle solution.

La vue temporelle, elle, utilise en plus un résolveur pour maintenir le placement spatial des objets dans sa vue. En cas de déplacement, la vue temporelle fait appel à son résolveur local pour propager les déformations, et une fois l'opération d'édition réalisée, elle propage le résultat sur le document de référence.

8 Evaluation des résolveurs de contraintes

Maintenant que nous avons décrit dans quelles conditions sont utilisés les résolveurs dans Kaomi, nous allons présenter les résultats de travaux qui ont permis d'évaluer les différents types de résolveurs dans notre contexte d'utilisation.

8.1 Résolveurs de contraintes évalués

Le but de cette étude n'est pas de comparer tous les résolveurs de contraintes existants, mais d'avoir une base de comparaison entre les différentes techniques de résolution. Le choix des résolveurs étudiés a été fait en fonction de deux critères :

- La représentativité de l'approche.
- Le langage d'implémentation (Java) de notre boîte à outils, le langage d'implémentation des résolveurs a été choisi de manière à simplifier leur intégration dans la boîte à outils.

Nous avons évalué sept résolveurs différents, cinq d'entre eux ont été développés par des universités ou des centres de recherche publics dans un contexte général, les deux autres ont été développés de manière spécifique à notre système.

8.1.1 Résolveurs globaux évalués

Les trois résolveurs à base de mécanismes globaux que nous avons évalués sont Cassowary, jSolver et Maple [Maple98].
Les deux premiers résolveurs ont été présentés dans la section 6. Nous rappellerons brièvement leur caractéristiques.

Cassowary :
Cassowary est un résolveur de contraintes pour les expressions linéaires. Il a déjà est utilisé pour calculer le placement spatial d'objets [Badros98], et il sert de base à d'autres résolveurs de contraintes tels que Qoca [Marriott98]. Son pouvoir d'expression couvre nos besoins (excepté la boîte minimale englobante), et il est possible d'orienter la recherche de solution. Nous avons utilisé les contraintes hiérarchiques pour réaliser cette résolution. De ces faits, il nous a semblé pertinent d'évaluer ce résolveur.

jSolver :
A la différence de Cassowary, jSolver est un résolveur énumératif, c'est-à-dire qu'il évalue toutes les combinaisons possibles de valeurs des variables. Cependant, il permet à l'utilisateur de définir ses propres heuristiques pour l'orientation de la résolution. Ainsi, nous avons pu définir une politique pour l'ordre d'évaluation des variables ainsi que sur le choix des valeurs. Pour définir ces heuristiques, nous nous sommes basés sur les méthodes définies dans la section 5.3.
De plus, pour améliorer ses performances, jSolver nous donne la possibilité de calculer un plan et de le conserver pour une succession d'opérations d'édition similaires.
De ce point de vue, et du fait qu'il représente une approche complètement différente de Cassowary, il nous a semblé judicieux d'évaluer ce genre d'approches, qui de plus, grâce au contrôle qu'elle offre sur la recherche de solution semble très intéressante dans notre contexte. On peut noter cependant qu'une des difficultés de ces approches est qu'elles manipulent explicitement l'intervalle de chacun des objets. Par exemple, si nous avons deux objets dont les intervalles de durée sont compris entre 9 et 10 minutes, cela laisse un choix de 60000 valeurs possibles, car nous travaillons en millisecondes.

Maple :
Maple ([Maple98]) a une place un peu particulière dans notre analyse : il est considéré comme un résolveur de référence. C'est-à-dire que les performances temporelles de ce résolveur ne nous intéresseront pas. Il servira seulement à calculer la meilleure solution. De plus, ce résolveur est non linéaire, il nous permet donc de manipuler des concepts comme la boîte minimale englobante. Nous l'avons aussi utilisé pour mesurer la qualité des solutions fournies par les autres résolveurs.

8.1.2 Résolveurs locaux évalués

La seconde catégorie de résolveurs que nous allons expérimenter est celle des résolveurs locaux. Ce type de résolveurs semble intéressant dans notre contexte du fait des bonnes performances temporelles qui les caractérisent. Comme représentant de cette catégorie de résolveurs nous avons choisi d'utiliser Deltablue.

Une des difficultés avec ce genre de résolveur est leur incomplétude. Cette incomplétude est à deux niveaux :

- Incapacité de traiter des graphes de contraintes cycliques (dès que nous avons des relations hiérarchiques nous avons des graphes cycliques (voir Figure V-18)). En effet, chaque objet est dans la boîte englobante définie par son père dans la hiérarchie. À cause de cette relation, et des relations qui lient les variables de chaque objet, nous avons systématiquement des cycles. Dans l'exemple de la Figure V-18 on peut voir le graphe de contraintes associé à la relation A inside B.
- Incapacité à garantir que le résolveur trouvera une solution s'il en existe une, même pour des graphes acycliques. Par exemple, si l'orientation du résolveur l'emmène vers une mauvaise piste.

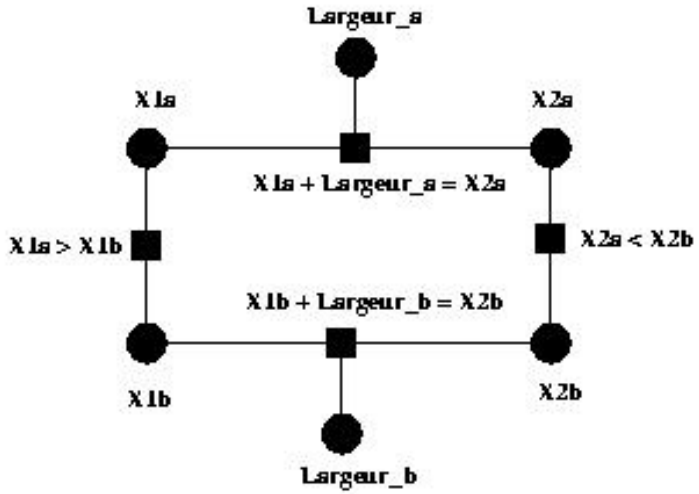


Figure V-18 : Cycle dans le graphe de contraintes

De manière à contourner ces limites, il est donc nécessaire d'effectuer un prétraitement pour éliminer ces cycles quand cela est possible, ou de limiter les déplacements permis à l'auteur, par exemple, en interdisant à un objet de modifier son composite englobant. En effet, cela se traduit par la suppression des relations père/fils dans le graphe de contraintes et le système de gestion de contraintes restreindra les manipulations de l'auteur de manière à ne pas déformer l'objet père (7.3.2). Par exemple, lorsque l'auteur modifie la variable X2a, on peut supprimer la contrainte qui lie X2a et X2b pour casser le cycle.

Le mécanisme de résolution propagera la déformation de X2a sur le graphe de contraintes. Cependant, le mécanisme de résolution peut choisir de modifier la largeur de B pour satisfaire l'ensemble de contraintes. Cette modification est pertinente du fait qu'il n'y a plus la contrainte (X2a < X2b).

Il faudra donc orienter le résolveur (en ajoutant des poids sur les contraintes) de telle manière qu'il satisfasse la contrainte que nous avons enlevée pour supprimer le cycle.

Ce mécanisme de suppression des cycles qui doit être recalculé à chaque opération d'édition (car il est dépendant de la variable affectée) est basé sur le travail de Laurent Carcone [Carcone97] lors de l'expérimentation. Le temps nécessaire pour résoudre ces cycles sera compté dans le temps de résolution.

8.1.3 Résolveurs ad hoc évalués

Comme nous avons pu le voir dans le chapitre précédent deux résolveurs ont été développés de manière spécifique pour notre utilisation (HPC2, AVT). Nous évaluerons leurs performances et leurs qualités par rapport aux autres résolveurs.

On rappelle que l'algorithme de la vue temporelle nécessite la connaissance des domaines de validité des variables, et de ce fait impose que le résolveur de contraintes utilisé pendant les phases d'ajout / retrait d'objet ou de relation maintienne ces informations.

De plus, nous avons défini un résolveur, le résolveur nul, il propage les valeurs sans calculer de nouvelles valeurs ni s'assurer de la cohérence du document. Cela nous permettra d'isoler le temps pris effectivement par le résolveur du temps mis par le système pour prendre en compte la modification et la propager.

8.1.4 Orientation des résolveurs de contraintes

Pour tous les résolveurs de contraintes supportant la définition de contraintes hiérarchiques nous avons utilisé le mécanisme suivant pour orienter la rechercher. La définition des poids sur les contraintes, et l'ajout de contraintes pour orienter la recherche nécessite un prétraitement pour chaque opération d'édition.

Par exemple, pour orienter la résolution du graphe de contraintes de la section 8.1.2 il faut :

- Rajouter une contrainte de poids de 5 qui indique que la variable Largeur_a est constante ceci afin de favoriser la modification de X1a par rapport à Largeur_a.
- Définir un poids de 3 sur la contrainte X1b + largeur_b = X2b et un poids de 4 sur la contrainte X1a=X1b. Cela aura pour effet de favoriser la modification de X2a par rapport à X2b.
- Rajouter une contrainte de poids de 5 qui indique que la variable Largeur_b est constante ceci afin de favoriser la modification de X2b par rapport à Largeur_b.

Notons qu'entre deux opérations d'édition, la politique d'orientation reste globalement la même et ne nécessite que quelques modifications locales. Le temps de prétraitement est comptabilisé dans le temps de résolution.

8.2 Expérimentation

Les tests ont été réalisés sur un Pentium II, 400 MHz .

8.2.1 Description des tests d'évaluation

Pendant cette phase d'évaluation, notre intérêt s'est porté sur trois points :

- Le temps mis pour calculer la nouvelle solution.
- La qualité de la nouvelle solution en fonction de la précédente.
- La capacité du résolveur à supporter le facteur d'échelle.

Pour réaliser ces tests on a généré deux groupes de documents, le premier contenait 100 objets par document (Figure V-19) et le second 1000 objets (Figure V-20).

Chaque session test est composée des étapes suivantes :

- Ouvrir le document et mesure du temps nécessaire au résolveur pour calculer une solution initiale à partir du jeu de contraintes et de l'ensemble des variables.
- Edition:
  - Ajout ou retrait d'un objet.
  - Ajout ou retrait d'une relation temporelle ou spatiale.
  - Modification d'un attribut spatial ou temporel par manipulation directe, c'est-à-dire que l'auteur déplace ou retaille l'objet à l'écran dans la vue de présentation ou dans la vue temporelle.

Pour chacun des résolveurs nous avons utilisé une des deux méthodes (heuristique ou contraintes pondérées) pour orienter la recherche de solution.

Le symbole L sera utilisé quand le résolveur n'aura pas été capable de trouver une solution, ou qu'il aura généré une erreur due, par exemple, à un problème de

mémoire. Le symbole *Non* signifie que le résolveur est incapable de traiter l'opération d'édition. Dans la Figure V-21, nous présentons les résultats qualitatifs, le critère de qualité a été calculé en mesurant la distance entre la solution générée par Maple et celle du résolveur que l'on étudie. Les paramètres utilisés pour cette fonction étaient : a = b = x = d = e = 0.2 (voir 5.3.1).

+,- : signifie ajout / retrait d'objet ou de relation ;
Dep. : signifie déplacement ;
Ret. : signifie retaillage.

Résolveur		**Ouverture**	**Edition**									
			Objets		Relations				Attributs			
			+	-	Spatiale		Temporelle		Spatial		Temporel	
					+	-	+	-	Dep.	Ret.	Dep.	Ret.
Globaux	Cassowary	4.4s	0.4s	0.2s	0.6s	0.3s	0.4s	0.2s	0.5s	0.5s	0.4s	0.4s
	JSolver	4.5s	0.5s	4.5s	1.2s	4.5s	1s	4.5s	4.5s	4.5s	4.5s	4.5s
	Maple	18mn	18mn	18mn	18mn	18mn	18mn	18mn	18mn	18mn	18mn	18mn
Local	Deltablue	4.5s	0.2s	0.1s	0.5s	0.3s	0.4s	0.3s	0.2s	0.2s	0.2s	0.2s
Nul		3s	0.1s	0.1s	0.1s	0.1s	0.1s	0.1s	0.05s	0.05s	0.05s	0.05s
Ad'hoc	HPC2	20.2s	0.15s	0.15	Non	Non	1s	20.2s	Non	Non	12s	12s
	AVT	Non	Non	Non	Non	Non	Non	Non	Non	Non	0.1s	0.1s

Figure V-19 : Résultats quantitatifs pour 100 objets

Résolveur		**Ouverture**	**Edition**									
			Objets		Relations				Attributs			
			+	-	Spatiale		Temporelle		Spatial		Temporel	
					+	-	+	-	Dep.	Ret.	Dep.	Ret.
Global	Cassowary	192s	0.7s	0.2s	5s	0.3s	4s	0.2s	3.5s	3.5s	2.3s	2.3s
	Jsolver	L	L	L	L	L	L	L	L	L	L	L
	Maple	> 1h	> 1h	> 1h	> 1h	> 1h	> 1h	> 1h	> 1h	> 1h	> 1h	> 1h
Local	Deltablue	11.4s	0.4s	0.2s	1s	0.3s	1s	0.3s	0.5s	0.5s	0.5s	0.5s
Nul		8s	0.1s	0.1s	0.1s	0.1s	0.1s	0.1s	0.05s	0.05s	0.05s	0.05s
Ad'hoc	HPC2	L	L	L	L	L	L	L	L	L	L	L
	AVT	Non	Non	Non	Non	Non	Non	Non	Non	Non	0.1s	0.1s

Figure V-20 : Résultats quantitatifs pour 1000 objets

Dans la Figure V-21, nous présentons les résultats qualitatifs. On peut noter qu'une valeur de qualité de 0 indique une solution parfaite en fonction du critère défini, une valeur supérieure à 0.2 indique une solution non satisfaisante.

Résolveur		ouverture	Edition									
			Objets		Relations				Attributs			
			+	-	Spatiale		Temporelle		Spatial		Temporel	
					+	-	+	-	Dep.	Ret.	Dep.	Ret.
Globaux	Cassowary	0.1	0	0	0.1	0	0.1	0	0.05	0.05	0.05	0.05
	jSolver	0.1	0	0	0.05	L	0.05	L	0	0	0	0
	Maple	0	0	0	0	0	0	0	0	0	0	0
Local	Deltablue	0.3	0	0	0.3	0	0.3	0	0.4	0.4	0.4	0.4
Od'hoc	HPC2	0.1	0	0	Non	Non	0	0	Non	Non	0	0
	AVT	Non	Non	Non	Non	Non	Non	Non	Non	Non	0.05	0.05

Figure V-21 : Résultats qualitatifs pour 1000 objets

8.2.2 Analyse des résultats de l'évaluation

Les résultats quantitatifs et qualitatifs montrent que les performances dépendent de l'opération d'édition réalisée. En effet, les opérations révèlent les capacités intrinsèques des résolveurs. Par exemple, de bonnes performances temporelles sont obtenues avec HPC2, lors de l'ajout de relations, du fait qu'il est capable de maintenir les domaines minimaux sur les intervalles, mais il a de très mauvaises performances temporelles lors du retrait de relations. Cela est dû au fait qu'il doit reconsidérer à nouveau l'ensemble des contraintes, car cet algorithme n'est pas adapté au relâchement des intervalles minimaux. On remarque que jSolver a le même comportement.

Le meilleur résolveur en terme de temps de réponse est Deltablue, cependant les limites que nous avons décrites dans la section précédente sont trop contraignantes dans un contexte d'édition, ce qui se remarque dans les résultats qualitatifs. Le résolveur qui offre donc le meilleur compromis entre les critères de performance et de qualité est Cassowary.

Les résultats montrent également la différence de sensibilité des résolveurs face au facteur d'échelle :

- HPC2 atteint vite ses limites lorsque le nombre d'objets manipulés devient important. Cette limite est liée à la manipulation d'un graphe complet par l'algorithme.
- jSolver atteint aussi rapidement ses limites du fait de la taille des domaines manipulés.

L'algorithme AVT nécessite les domaines minimaux pour être capable de calculer une nouvelle solution. Ce calcul est complexe et coûteux en temps. Cet algorithme ne peut donc pas être utilisé lors de l'ajout ou du retrait d'objet ou de relation. Cependant, cet algorithme est très puissant pour calculer le déplacement et le retaillage d'objets (avec les restrictions décrites dans le chapitre précédant). Il est donc particulièrement adapté aux manipulations de navigation dans la vue temporelle et d'exécution.

Un point important à souligner est que les résolveurs les plus performants sont ceux qui ont les plus mauvaises performances qualitatives, car n'offrant pas de mécanismes efficaces de contrôle pour orienter la recherche de solution.

8.3 Proposition d'un algorithme polymorphe

Les résultats précédents montrent que le meilleur algorithme dépend de l'opération d'édition. De cette analyse, nous avons décidé de combiner plusieurs résolveurs de contraintes de manière à utiliser le plus adapté pour chacune des opérations d'édition.

L'idée de combiner plusieurs résolveurs de contraintes a déjà été expérimentée dans différents contextes. On peut citer les résolveurs SkyBlue [Sannella95], Ultraviolet [Borning98] et DETAIL [Hosobe96]. Lors de la combinaison de différents résolveurs se posent plusieurs problèmes, on peut citer :

- Le partage ou la synchronisation des différentes structures de données ;
- Les critères de chargement des résolveurs ;
- Le pilotage des résolveurs. Le pilotage peut être *externe* dans ce cas c'est un module qui gère la coopération entre les résolveurs, ou *hiérarchique* et dans ce cas c'est un des résolveur qui contrôle les autres résolveurs.

Les résolveurs Skyblue et Ultraviolet fonctionnent sur le même mécanisme. Un résolveur principal, qui travaille sur le graphe de contraintes résout tant qu'il le peut le graphe de contraintes. Lorsqu'il trouve une sous-partie du graphe, avec une topologie telle qu'il ne peut la traiter (ou la traiter de manière efficace), il fait appel à un résolveur secondaire pour traiter cette partie de graphe. Les résolveurs secondaires sont en général moins performants que les résolveurs primaires, mais ils permettent d'étendre leur expressivité. Dans notre contexte, la topologie de nos graphes de contraintes est telle, que ces résolveurs feraient constamment appel aux résolveurs secondaires, et perdraient ainsi tous les avantages du résolveur primaire.

Le résolveur DETAIL est un résolveur multicouches, c'est à dire qu'il est composé d'un résolveur principal et de sous-résolveurs. Le résolveur principal produit le graphe de contraintes et applique dessus une propagation locale. Au cours de cette propagation il divise le graphe en cellules correspondant à des zones du graphe facilement calculables. Chacune de ces cellules sera donnée à un sous-résolveur qui calculera une valuation pour chacune des variables de la cellule. Le résolveur choisira un sous-résolveur optimisé pour la topologie et les contraintes de la cellule. DETAIL ne traite pas les inégalités et obtient des performances temporelles proches de Deltablue.

L'algorithme polymorphe que nous avons utilisé est une combinaison de Deltablue et de Cassowary. La coopération entre ces deux algorithmes ne se fait pas pour la résolution d'un problème, mais une entité de pilotage et de contrôle externe aux résolveurs choisit, en fonction du problème, quel résolveur elle utilisera. Nous utilisons par exemple Deltablue lors de l'ouverture du document ainsi que pour l'ajout d'objet. Nous utilisons Cassowary pour toutes les autres opérations d'édition du fait qu'il est moins restrictif que Deltablue. L'utilisation de Deltablue, lors de l'étape d'ouverture n'est pas trop restrictive, en effet, à cette étape, il n'est pas important de manipuler toutes les contraintes, on peut retirer celles qui introduisent des informations redondantes ([Carcone97]).

La principale difficulté de cette coopération sera la synchronisation des différentes structures de données (celle de Kaomi, celle de Deltablue et celle de Cassowary). Cette synchronisation se fait grâce à la structure de données de Kaomi. Avant de résoudre un problème, le résolveur vérifie que ses données sont cohérentes par rapport à celles de Kaomi, et à la fin de la résolution, le résolveur met les structures de données de Kaomi à jour.

Dans la Table 4, on peut voir que l'algorithme polymorphe n'est pas celui qui obtient les meilleurs performances temporelles, cela est lié au fait que le temps nécessaire pour prendre en compte une opération d'édition est composée de deux parties : le temps pour construire les informations du résolveur et le temps nécessaire pour calculer la nouvelle solution. Si nous utilisons les deux algorithmes, il faut aussi compter un temps pour mettre à jour les informations du résolveur qui n'est pas utilisé.

Malgré cela, ce résolveur semble très prometteur et c'est lui qui offre globalement le meilleur rapport temps/qualité.

Ouv.	Editon									
	Objets		Relation				Attributs			
	+	-	Spatial		Temporel		Spatial		Temporel	
			+	-	+	-	Dep.	Ret.	Dep.	Ret.
100 objets / document	5.5s	0.3s	0.2s	0.7s	0.4s	0.5s	0.3s	0.5s	0.4s	0.4s
1000 objets / document	13.2s	0.6s	0.3s	2s	0.4s	1.7s	0.4s	0.7s	2.4s	2.5s

Figure V-22 : Résultats de l'algorithme polymorphe

9. Apports et limites des contraintes pour l'édition directe

Dans Kaomi, nous utilisons actuellement une implémentation de l'algorithme polymorphe.

L'utilisation des technologies contraintes a permis de faciliter la tâche de conception de la boîte à outils et, comme nous le verrons par la suite, de faciliter la tâche de l'auteur dans les différents environnements auteur construits à l'aide de la boîte à outils.

Dans la réalisation des services de Kaomi, les contraintes ont permis de :

- Faciliter la tâche de conception et les possibilités d'extension en généralisant la spécification des contraintes à résoudre.
- Mettre en commun les mécanismes de vérification de formatage pour les différents formats de fichiers.
- Rendre l'extension des relations accessibles par l'auteur sans pour autant remettre en cause les mécanismes de résolution.
- Permettre la mise en place d'une politique d'orientation générale pour le formatage ainsi que la possibilité de la modifier facilement.

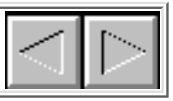
D'un point de vue édition, les contraintes ont permis de :

- Réaliser une édition incrémentale avec un formatage dynamique lors des opérations d'édition.
- Réaliser une édition par manipulation directe dans la vue temporelle et spatiale, avec maintien des relations et vérification de la cohérence en cours de manipulation.
- Faciliter l'expression du scénario spatial et temporel grâce au formalisme à base de contraintes offert par Kaomi.

Cependant, malgré ces apports, l'utilisation des contraintes présente encore des limites :

- Le choix du résolveur influe sur la capacité du système à assurer la cohérence du document. En effet, l'utilisation d'un algorithme de résolution non complet, de type Deltablue nous oblige à réaliser une phase de vérification de la cohérence explicite.
- Les performances temporelles des résolveurs sont inversement proportionnelles au nombre d'objets manipulés. Ces techniques sont encore difficilement utilisables pour de gros documents (présentation contenant 10000 objets).
- Traitement des objets imprédictifs, en effet l'intégration d'objets imprédictifs dans les relations d'égalité et d'implication, pose de nouveaux problèmes du

fait que l'on ne peut pas formater statiquement le document. Une méthode consiste à définir une fin prévue et une fin effective sur chacun des objets. La difficulté est que le système ne peut et ne doit pas affecter de valeur à ces variables du fait que leurs valeurs ne seront connues que lors de l'exécution. La fin prévue correspond à une durée calculée statiquement, et la fin effective correspond à celle qui aura lieu dynamiquement. Dans le cas d'objets contrôlables interrompus par un objet incontrôlable, ces deux valeurs de fin sont différentes. Statiquement, il faut assurer qu'il existe au moins une solution. De plus il faut vérifier que toutes les exécutions seront cohérentes ce qui devient une tâche complexe et non réalisable en temps polynomial [Mackworth85].



Chapitre VI : Expérimentations, évaluations et extensions de Kaomi

1. Introduction

Comme nous l'avons introduit précédemment, un des objectifs de la thèse est de valider la boîte à outils Kaomi au travers de la réalisation de plusieurs environnements auteur reposant sur des formalismes suffisamment différents pour montrer les possibilités d'utilisation de la boîte à outils.

Nous présenterons dans un premier temps les outils auteur de documents multimédias construits à partir de formalismes relationnels et événementiels (section 2). Dans un deuxième temps, nous présenterons deux expériences d'utilisation de la boîte à outils différentes des précédentes puisqu'elles visent à construire d'une part un éditeur de workflow (section 3), et d'autre part à utiliser Kaomi dans un contexte de base documentaire (section 4).

Ce travail que j'ai initié tant d'un point de vue de la conception que de la réalisation a impliqué de nombreuses personnes au sein du projet Opéra, tant d'un point de vue programmation (ingénieurs, doctorants, stagiaires) que d'un point de vue recherche. De manière à bien identifier les apports et les contributions de chacun, la section 5 présentera un bilan quantitatif des différentes participations dans la boîte à outils ainsi que dans les différents environnements auteur construits grâce à elle. Nous ferons dans la section 6 un bilan des différentes expériences réalisées avec Kaomi. De ce bilan et de l'expérience acquise au cours de ces réalisations nous dégagerons un ensemble de propositions pour étendre cette boîte à outils (section 7).

2. Les outils auteur de documents multimédias réalisés avec Kaomi

Les environnements auteur de documents multimédias construits au-dessus de Kaomi permettent d'éditer des langages qui couvrent un large échantillon des formalismes de spécification de documents multimédias (voir Chapitre II section 3).

Les environnements auteur réalisés recouvrent :

- le formalisme relationnel :
  - à base de relations : Madeus-Editeur (section 2.1) ;
  - à base d'opérateurs : SMIL-Editeur [Navarro00] (section 2.2);
- le formalisme événementiel : MHML, illustré par la réalisation de l'environnement MHML-Editeur (section 2.3).

2.1 Réalisation de Madeus-Editeur

Madeus-Editeur est un environnement auteur de documents au format Madeus (voir Chapitre II section 3.4.2). Nous avons dû, dans un premier temps étendre, le parseur pour permettre la construction de la structure de données de Madeus, et enfin, nous avons rajouté deux fichiers de ressources, pour définir la sémantique des relations et des opérateurs du langage Madeus. La création de cet éditeur a été simplifiée car les formalismes d'édition proposés par Kaomi et Madeus-Editeur sont très proches.

La traduction des différentes relations de Madeus en rélems a servi à illustrer les mécanismes de Kaomi dans le chapitre IV, nous ne présenterons ici que la traduction de la relation spatiale "centrer" en rélems (Figure VI-1).

CentrerHorizontal (A,B)	A.Gauche + d1 = B.Gauche A.Droite ± d2 = B.Droite d1=d2
CentrerVertical (A,B)	A.Bas + d1 = B.Bas A.Haut ± d2 = B.Haut d1=d2

Figure VI-1 : Traduction des relations spatiales "centrer"

L'environnement ainsi réalisé permet à l'auteur d'éditer le document dans un ensemble de vues, chacune de ces vues étant synchronisées avec les autres. Les services offerts par Madeus tirent parti se ceux de la boîte à outils : cohérence, édition par manipulation directe, formatage.
L'utilisation des résolveurs de contraintes nous a permis notamment de permettre l'édition par manipulation directe dans la vue temporelle.
Lors de la sélection d'un objet dans la vue temporelle, le système visualise l'intervalle de déplacement possible pour l'objet ou l'intervalle de retaillage possible. De plus, le système met en évidence les objets qui seront impliqués dans la déformation de l'objet (à cause des relations) par un changement de couleur, que ce soit lors un déplacement ou d'un retaillage.

Dans l'exemple de la Figure VI-2 l'auteur a sélectionné l'objet Photo4, le système visualise l'intervalle dans lequel l'auteur peut déplacer cet objet. Par exemple, le déplacement vers la gauche de l'objet Photo4 est limité par le fait que cet objet possède une relation *meet* avec l'objet Photo3. Lorsque l'auteur déplace l'objet (Figure VI-3) la vue se met à jour de manière continue. Le calcul des nouvelles positions se fait grâce aux résolveurs de contraintes. Les objets impliqués par la modification de l'auteur sont les objets Photo2 et Photo3. Le système interdira à l'auteur tout déplacement en dehors de l'intervalle permis. L'auteur peut aussi retailler son objet (Figure VI-4), comme pour un déplacement, la vue se mettra à jour en temps réel et interdira les déplacements non permis.

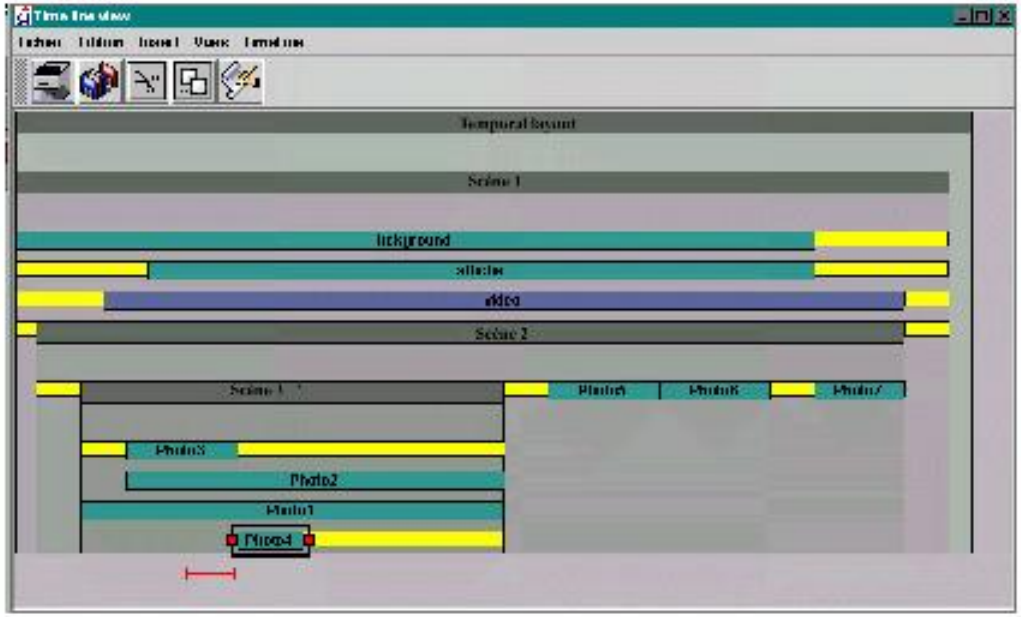


Figure VI-2 : Sélection d'un objet dans la vue temporelle de Madeus Editeur

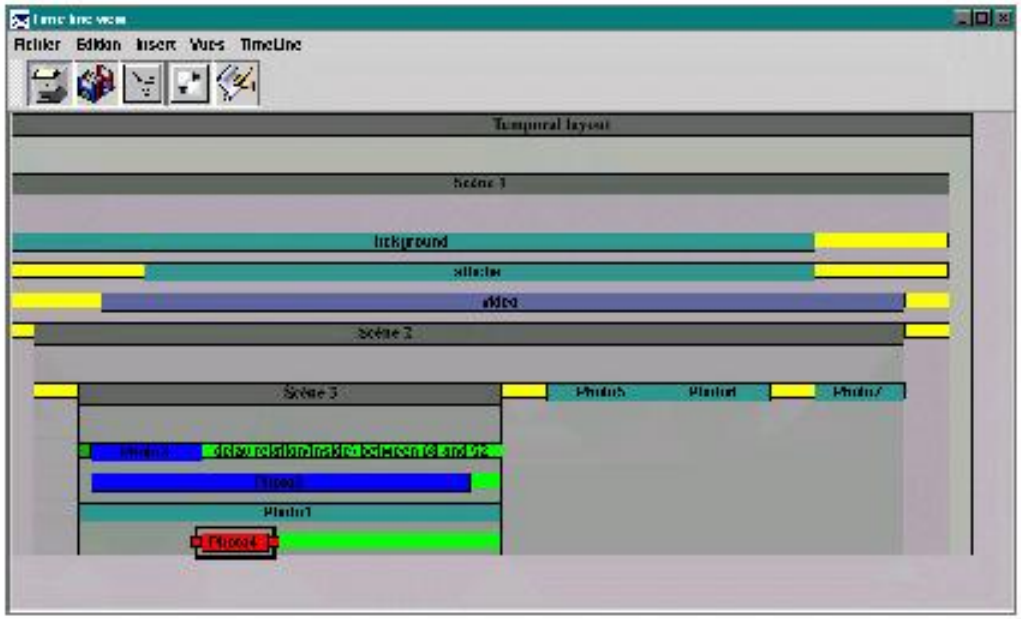


Figure VI-3 : Déplacement d'un objet dans la vue temporelle de Madeus Editeur

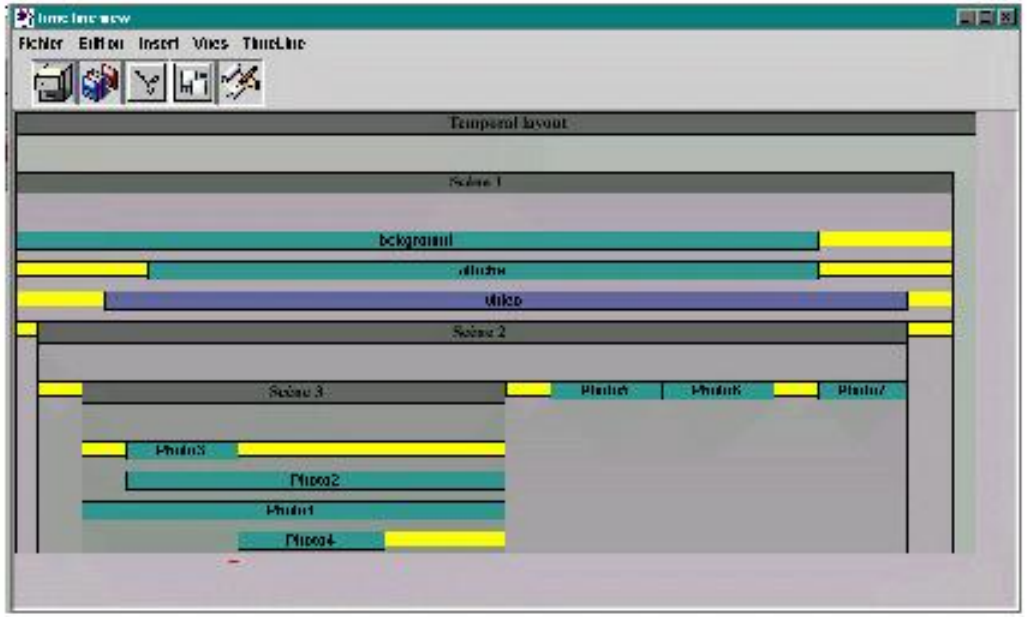


Figure VI-4 : Retaillage d'un objet dans la vue temporelle de Madeus Editeur

Par rapport à la version précédente de Madeus (Madeus-97), ce sont essentiellement les services d'aide à l'édition qui ont été ajoutés. Parmi ces services, on peut noter :

- *L'édition incrémentale*, c'est-à-dire que lors de l'édition toutes les vues sont synchronisées et mises à jour après chaque opération d'édition.
- *La vue temporelle* manipulable, qui permet à l'auteur de modifier intéractivement les différents attributs temporels de ces éléments ainsi que de naviguer dans l'espace de solutions.
- *Le formatage*, qui permet de dégager l'auteur de la spécification précise de la durée de ces médias quand cela est possible.
- *La vérification de cohérence qualitative et quantitative*après chaque opération d'édition.

2.2 Réalisation de SMIL-Editeur

SMIL-Editeur [Jourdan 99a, Navarro2000] est un environnement auteur de documents au format SMIL. Pour cet éditeur, nous avons dans un premier temps réalisé les mêmes opérations que dans le cadre du langage Madeus, c'est-à-dire que nous avons traduit les relations SMIL en termes de rélems. Cette traduction est contextuelle, c'est-à-dire qu'un élément est traduit en ayant une connaissance de ces fils et des différents attributs positionnés sur ceux-ci. Dans la Figure VI-5 nous pouvons voir les traductions des principales relations définies dans SMIL.

<Par dur="12s"> <txt id="A" ../> <txt id="B" ../> </Par>	Par.Début = A.Début A.Début = B.Début Par.Fin => A.Fin Par.Fin => B.fin /* car nous ne connaissons pas la durée de A et B */
<Par endsync="first"> <txt id="A" dur="3s" /> <txt id="B" dur="8s"/> </Par>	Par.Début=A.Début A.Debut = B.Debut A.Fin Þ B.Fin A.Fin Þ Par.Fin /* car la durée de A est inférieur à la durée de B */
<Seq > <txt id="A" dur="3s" /> <txt id="B" dur="8s"/> </Seq>	Seq.Début = A.Début A.Fin = B. Début Seq.Fin Þ B.Fin
<Par > <txt id="A" dur="3s" begin="end (B)"/> <txt id="B" dur="8s"/> </Par>	Par.Début = B.début A.Début= B.Fin Par.Fin = A.Fin
<Par dur="12"> <txt id="A" dur="10" begin="5"/> <txt id="B" dur="16"/> </Par>	Par.Début=B.Début A.Début = B.Début A.Début D 5 Par.Fin Þ A.Fin Par.Fin Þ B.Fin

Figure VI-5 : Traduction des relations SMIL en rélems

Dans un deuxième temps, nous avons étendu les opérations d'édition permises dans Kaomi de manière à assurer la cohérence d'un ensemble de comportements propres à SMIL. Par exemple, lors de l'ajout d'un objet, nous lui affectons une région spatiale par défaut, de même, lors de la suppression d'une région, nous vérifions que cette région n'est utilisée par aucun objet.

De plus, dans cet éditeur, nous avons étendu l'expressivité du langage SMIL pour faciliter l'édition spatiale. Pour cela, nous avons donné la possibilité à l'auteur d'ajouter des relations spatiales entre les objets. Ces relations sont stockées dans le fichier source en utilisant le mécanisme des espaces de noms [XML99]. Ces relations sont donc retrouvées lors de la prochaine session d'édition. L'utilisation des espaces de noms rend inexploitable ces informations dans les autres outils d'édition.

Cet éditeur profite, comme Madeus-Editeur, des différentes fonctionnalités offertes par Kaomi. Parmi celles-ci, on peut noter la possibilité d'éditer directement le placement spatial et temporel dans les vues de présentation et temporelle, chose qui n'est pas permise actuellement par les autres outils d'édition de documents SMIL (voir Chapitre II section 4.1.2 et 4.3.2).

L'édition de la structure temporelle est facilitée par la vue hiérarchique (Figure VI-6). Cette vue permet à l'auteur d'ajouter facilement un ensemble de médias organisés temporellement, organisation que l'auteur ajustera par la suite dans la vue temporelle.

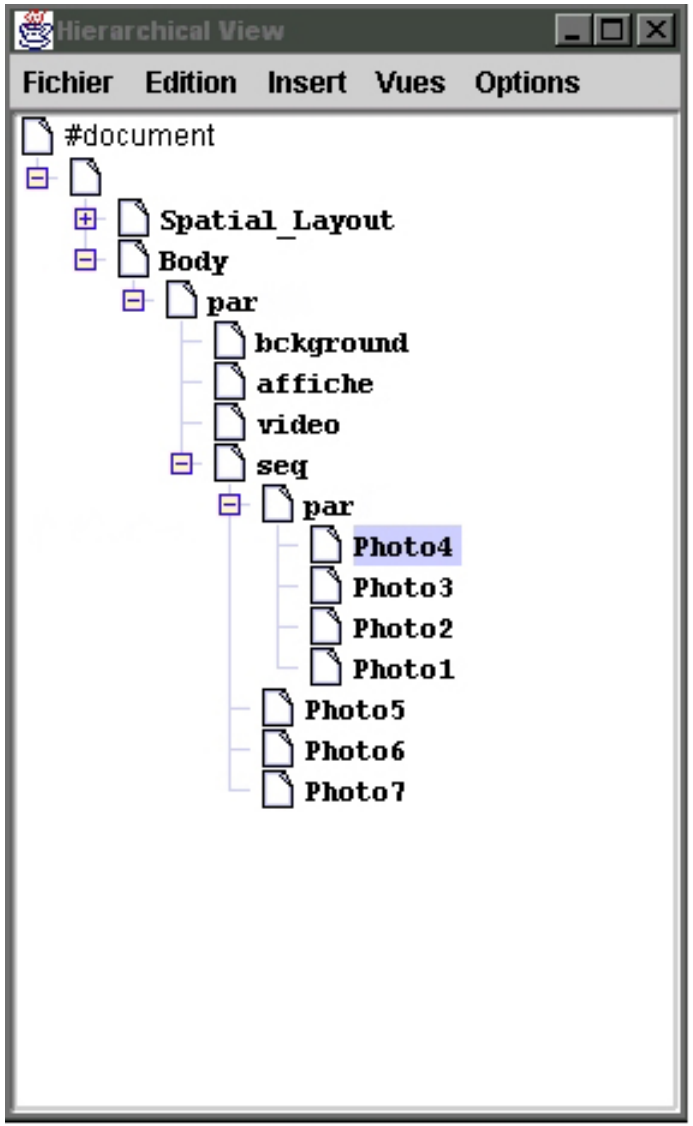


Figure VI-6 : Vue hiérarchique offerte par SMIL-Editeur

La vue temporelle permet de visualiser les différentes informations liées aux relations temporelles, mais aussi aux attributs temporels du langage SMIL. Dans la Figure VI-7, on peut voir les différentes informations temporelles présentées. On peut noter que les opérateurs de SMIL (*Par* et *Seq*) sont représentés par le mécanisme de boîtes englobantes fourni par Kaomi. Les attributs *begin* et *end* sont représentés explicitement par des délais reflétant les décalages temporels qu'impliquent les attributs (en jaune sur la figure). Enfin, les objets sont représentés par une boîte dont la longueur est proportionnelle à leur durée.

Les différentes manipulations permises sont le déplacement, le retaillage des objets et des composites. Ces fonctions sont directement issues de Kaomi.

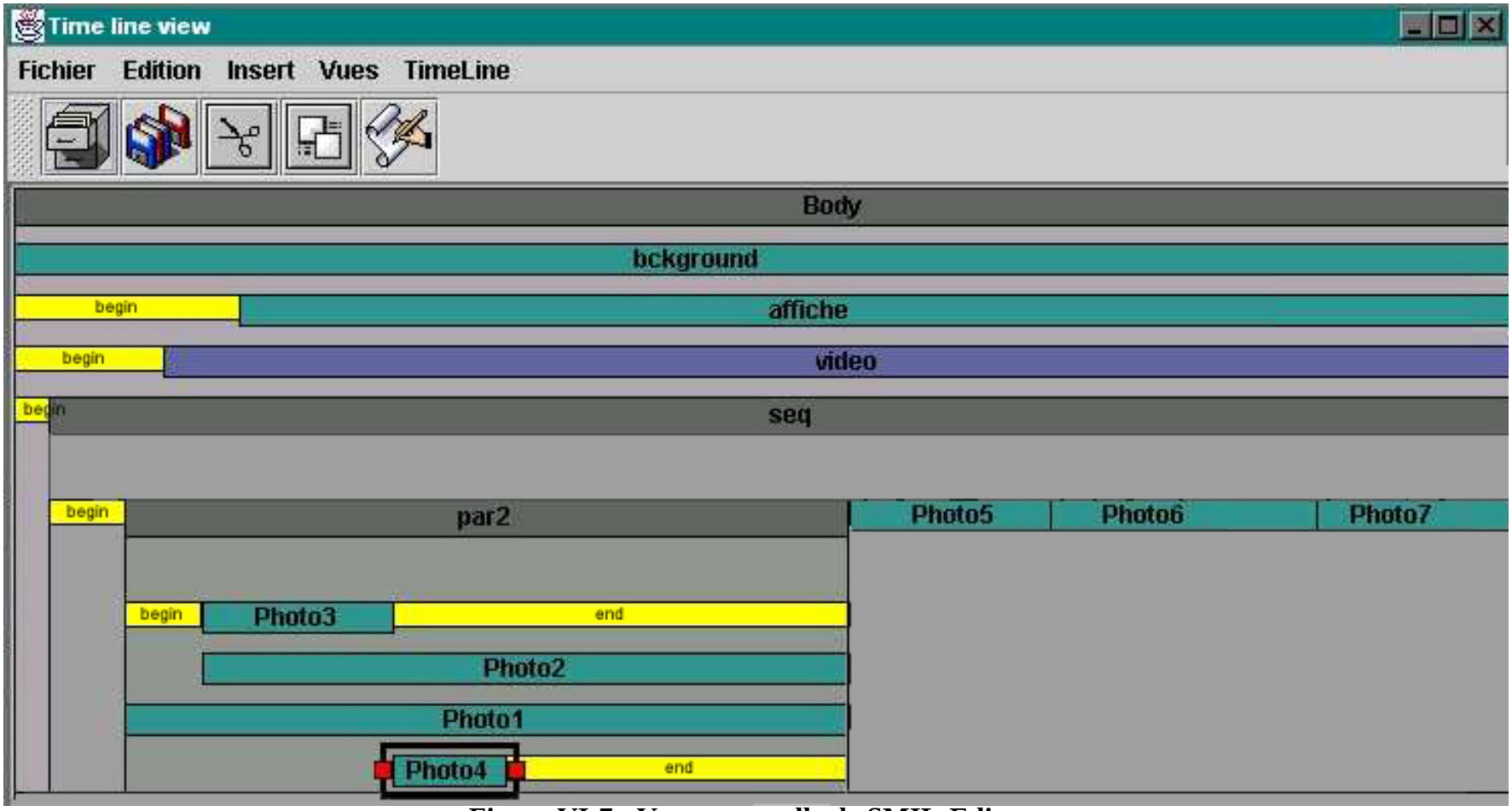


Figure VI-7 : Vue temporelle de SMIL-Editeur

Nous avons présenté dans le chapitre II des exemples d'environnement d'édition SMIL représentatif de ce qui existe aujourd'hui. Celui qui est le plus proche de notre éditeur SMIL-Editeur est certainement Grins (chapitre II section 4.3.2). Nous pensons cependant répondre plus complètement aux besoins des auteurs grâce aux fonctions suivantes :

- *L'édition incrémentale*, c'est-à-dire que lors de l'édition toutes les vues sont mises à jour.
- *La vue temporelle* manipulable, qui permet à l'auteur de modifier intéractivement les différents attributs temporels de ces éléments. Cette vue permet aussi de diminuer le nombre d'informations visualisées dans la vue temporelle grâce à l'ouverture/fermeture des éléments composites offerts par Kaomi.
- *La vue de présentation* qui permet à l'auteur d'éditer directement le placement spatial de ces objets.
- *L'intégration harmonieuse de deux formalismes d'édition* : celui du langage SMIL et celui de Kaomi pour le placement de relations spatiales et la définition des attributs dans un intervalle de valeurs.
- *Le formatage*, qui permet de calculer statiquement les instants de début et de fin des objets. Cela permet entre autre d'offrir une visualisation *a priori*

de l'enchaînement temporel défini dans le document.

- *Un premier niveau de vérification de cohérence qualitative*: nous vérifions qu'il n'y a pas de définition cyclique dans les attributs. Par exemple, lorsque la fin de l'objet A est définie relativement à la fin de l'objet B et réciproquement.

Néanmoins, l'outil proposé (contrairement à Grins) ne couvre pas complètement tous les traits du langage SMIL 1.0. L'opérateur *switch,* par exemple, n'est pas implémenté.

L'utilisation des techniques à base de contraintes a permis d'étendre facilement l'édition par manipulation directe dans les vues de présentation et temporelle pour le langage SMIL. De plus, ces techniques ont permis d'offrir un service de formatage qui permet de ne spécifier que partiellement les durées des objets de son document, laissant le soin au formateur de propager ces valeurs et de calculer une solution au document.

Dans la conclusion de cette thèse nous nous placerons par rapport à l'évolution de SMIL et notamment par rapport à la sortie de SMIL-2.0.

L'outil SMIL-Editeur a aujourd'hui servi de support à de nombreuses démonstrations que ce soit lors de conférences ou de réunions, néanmoins il manque une évaluation et un retour d'utilisateurs.

2.3 Réalisation de MHML-Editeur

L'éditeur MHML-Editeur, réalisé dans le cadre d'un contrat avec Alcatel, nous a permis d'expérimenter les modules liés à l'aspect imprédictif des médias et des événements.

Nous avons vu dans le chapitre II section 3.3 que la description des comportements temporel et spatial de MHML se fait via des événements. Parmi ceux-ci, certains sont prédictifs et donc traduits dans Kaomi sous forme de rélem, d'autres sont imprédictifs. Kaomi offre, de ce fait pour les événements prédictifs un service d'édition évolué avec manipulation dans les vues temporelle et de présentation. Par contre, même si Kaomi permet à l'auteur d'exprimer des comportements imprédictifs, elle n'offre pas de service d'assistance à l'édition (cohérence, manipulation directe).

Pour assurer un niveau acceptable de services d'édition, nous avons fait le choix de restreindre le langage MHML de manière à n'autoriser les événements imprédictifs que vers des îlots. Un *îlot* est une partie de document n'ayant pas d'événements imprédictifs entrant autres que ceux qui permettent de le démarrer et de l'arrêter.

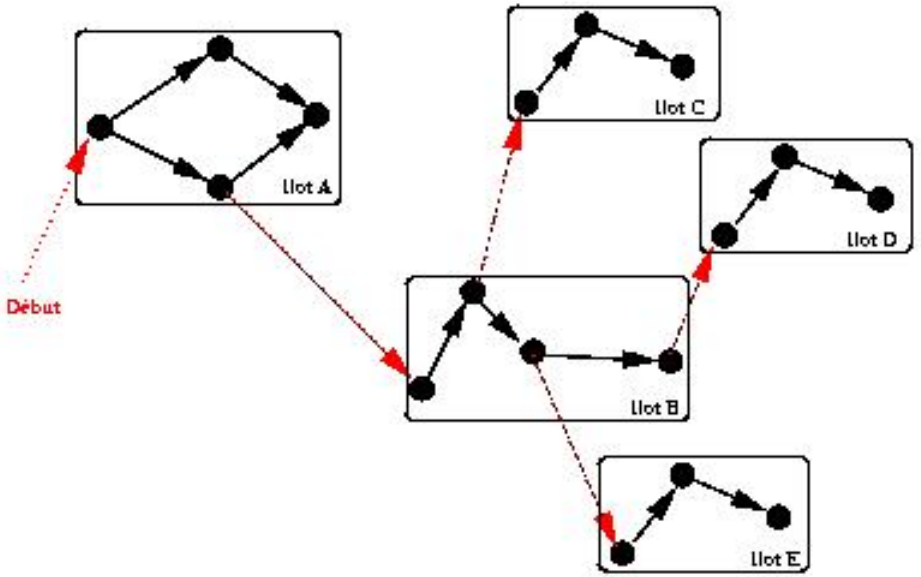


Figure VI-8 : Ensemble d'îlots d'un document MHML

Nous avons étendu Kaomi pour offrir une aide à l'édition de tels comportements. Cette édition se fait au travers d'une palette d'attributs qui permet d'éditer la structure des événements (Figure VI-9).

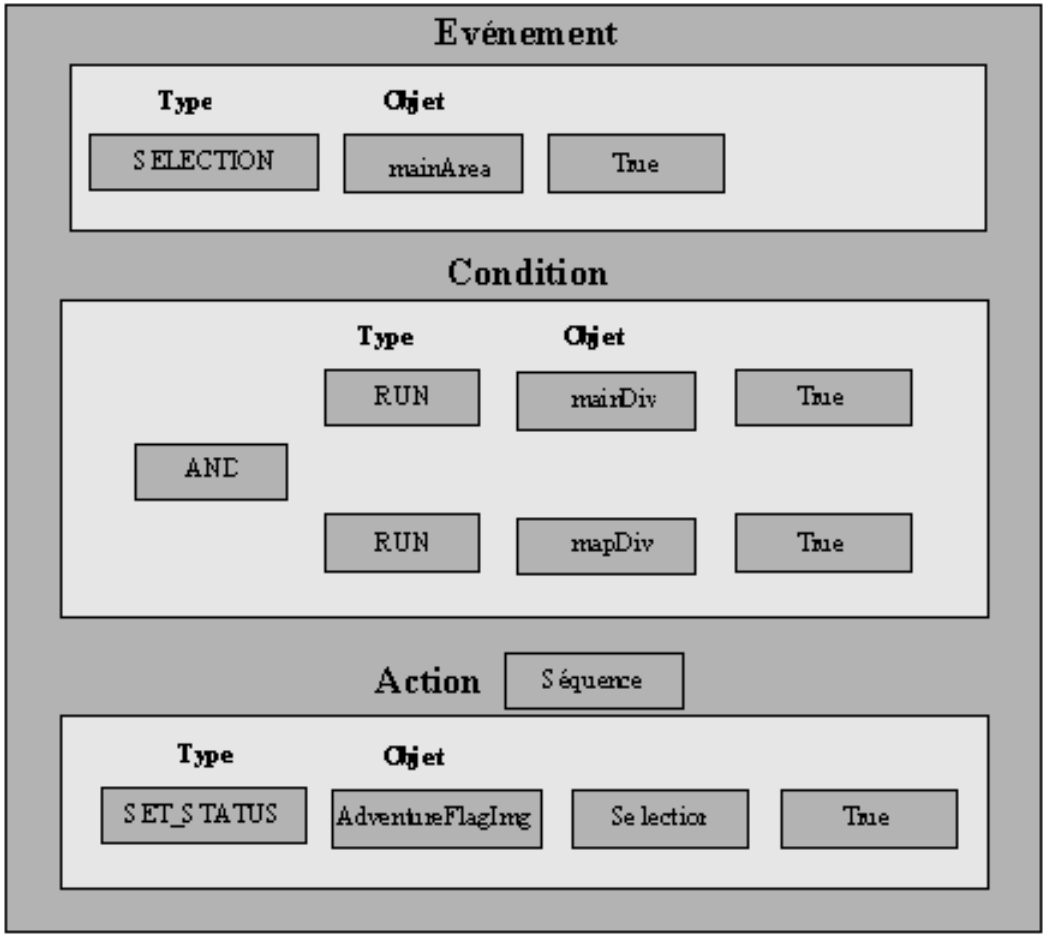


Figure VI-9 : Exemple de formulaire permettant le paramétrage d'événements

Par exemple, MHML-Editeur fournit un mécanisme de visualisation d'événements qui permet à l'auteur de percevoir des comportements imprédictibles, ainsi qu'un service de simulation qui permet d'obtenir des résultats d'exécutions possibles sans pour autant devoir exécuter le document (ce qui prendrait beaucoup plus de temps). Ce mécanisme de visualisation des événements repose sur un module de simulation qui permet de simuler des exécutions du document. Ce module a été réalisé au sein de Kaomi, permettant ainsi d'intégrer cette fonctionnalité dans d'autres outils auteur si le besoin s'en fait sentir. Ce module permettrait, par exemple, de simuler les comportements imprédictifs de SMIL 2.0.

Le choix qui a été fait permet de visualiser une solution possible parmi l'espace de solutions. La possibilité est donnée à l'auteur de choisir les événements qui ont lieu dans une boîte de dialogue, la date d'occurrence des événements étant choisie par le module de simulation. Dans la Figure VI-10, on peut voir un exemple de placement temporel avec quatre événements qui ont été déclenchés. Le premier est l'événement de début de document, les deux suivants ont été déclenchés, soit par une interaction utilisateur, soit par la fin d'un objet indéterministe et enfin le dernier est l'événement de fin de document qui dans ce cas a été déclenché par l'élément Body.

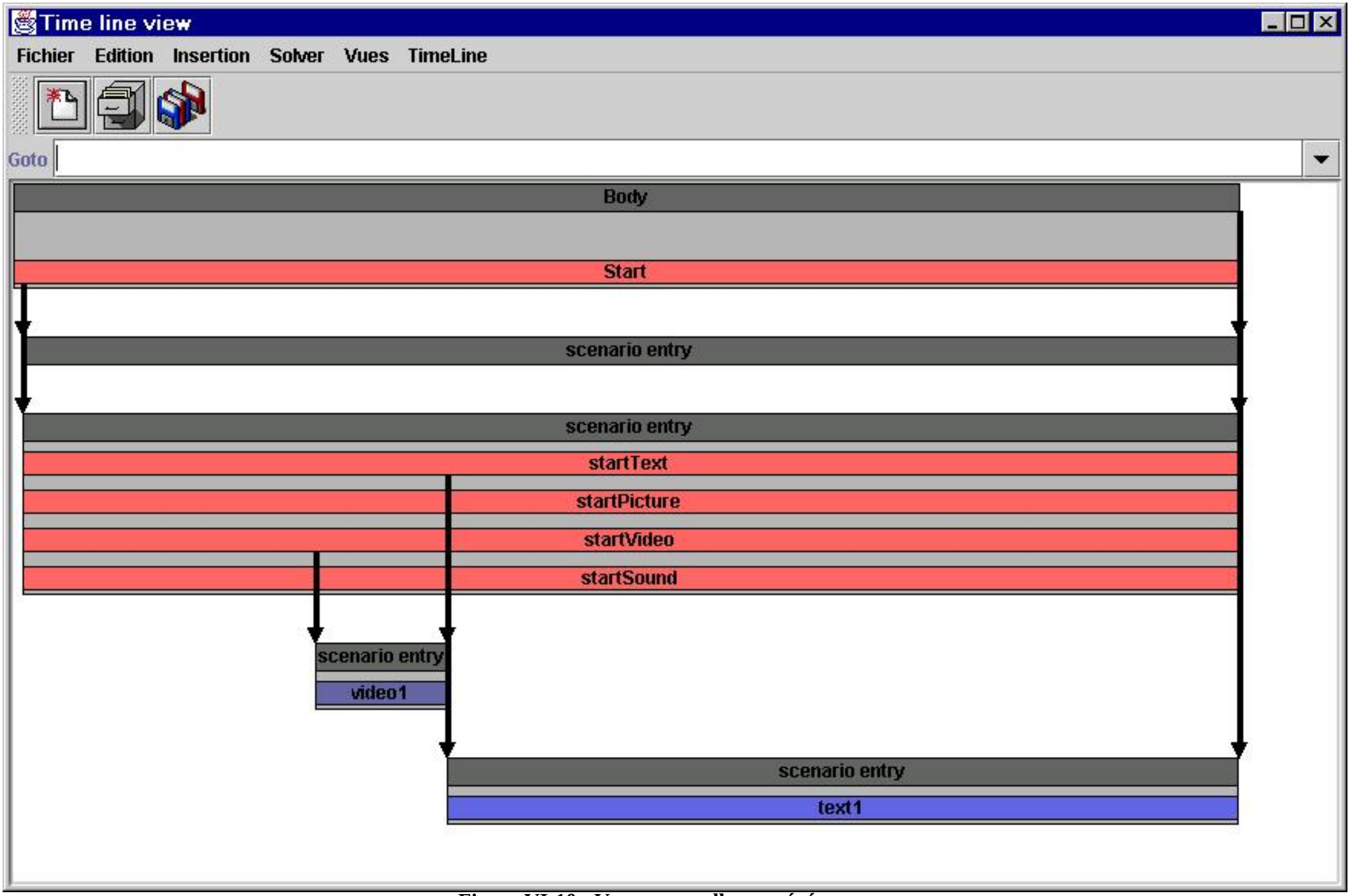


Figure VI-10 : Vue temporelle avec événements

À partir de cette représentation, l'auteur peut visualiser d'autres événements en les activant à partir de la boîte de dialogue. La vue s'adapte alors pour

prendre en compte l'occurrence de l'événement supplémentaire. Dans la Figure VI-11 l'auteur a cliqué sur StartVidéo pour déclencher un événement.

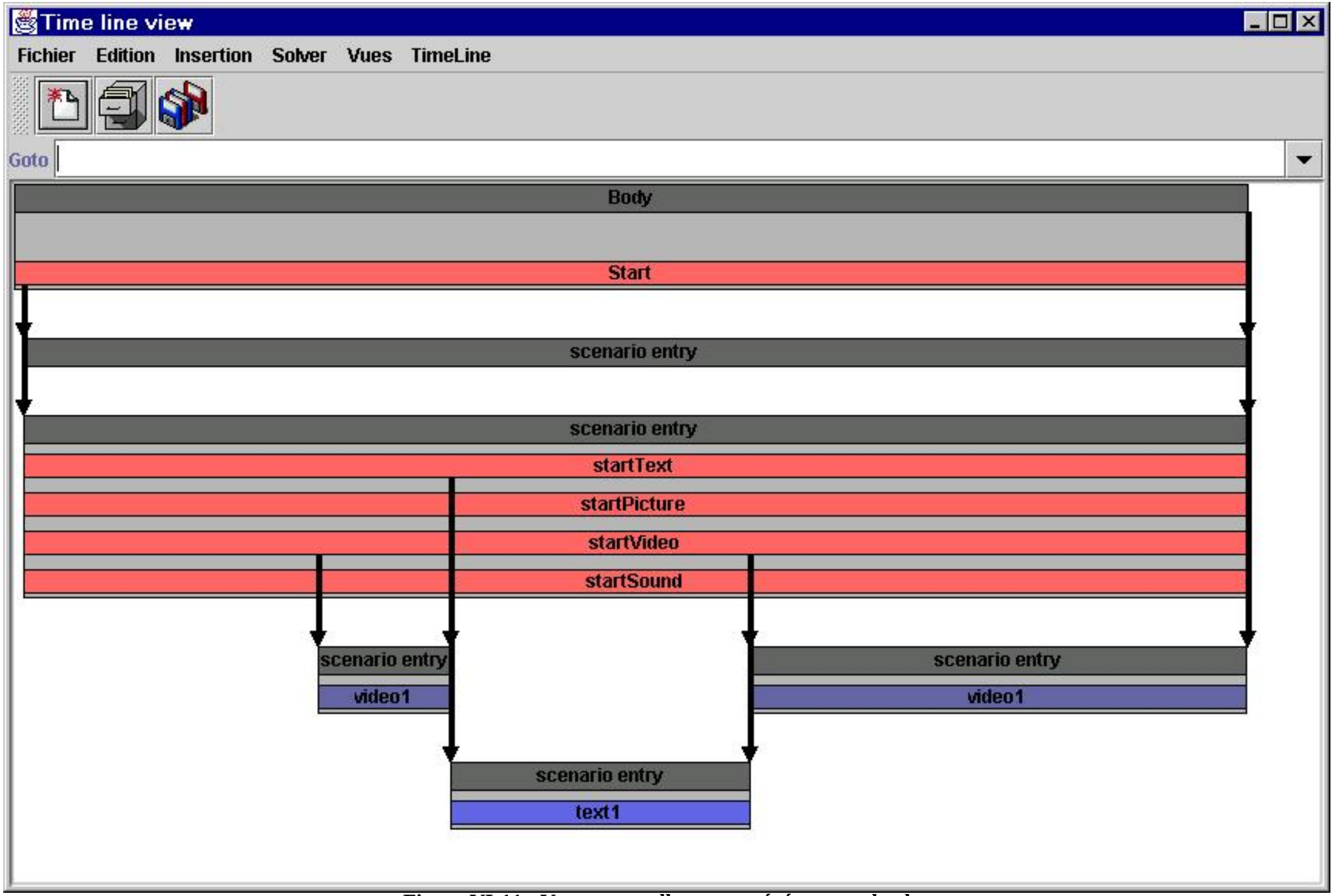


Figure VI-11 : Vue temporelle avec un événement de plus

Il serait intéressant d'étendre ce service en permettant à l'auteur de manipuler graphiquement les dates d'occurrence des événements.

Cet environnement peut être comparé à Lingo de Director (chapitre II section 4.3.1) de par l'expressivité qu'il permet à l'auteur. La différence essentielle vient de la facilité de spécification liée à l'utilisation de relations de haut niveau. Par rapport à Director, une autre différence essentielle est l'édition par manipulation directe avec maintien des relations (événements prédictifs) dans les vues temporelle et spatiale sur les parties prédictives du document.

L'environnement MHML-Editor permet à l'auteur de visualiser le résultat d'une exécution et de visualiser *a priori* plusieurs exécutions possibles dans différentes vues. Ces aides sont utiles à l'auteur de documents complexes pour visualiser le comportement de son document lors d'interactions avec le lecteur.

On peut noter, qu'il serait appréciable que des garanties puissent être fournies par le système auteur sans que l'auteur ne simule les différentes exécutions possibles (ce qui est impossible pour lui dans le cadre de documents imprédictifs) pour garantir certaines propriétés comme l'atteingnabilité de certaines parties de document, la durée minimum de présentation d'un média.

3. L'outil auteur de workflow réalisé avec Kaomi

Une expérience un peu différente a consisté à réaliser un outil auteur de workflow. Un workflow peut être défini comme un processus mettant en oeuvre un ensemble de tâches à réaliser. Chacune de ces tâches impliquant un ensemble de ressources humaines ou matérielles. Ces tâches ne sont pas indépendantes les unes des autres : le commencement d'une tâche T2 suppose par exemple la fin de la tâche T1. Il se peut aussi que pour des contraintes d'utilisation de ressources, les tâches T3 et T4 doivent se réaliser en séquence.

Les applications des workflows sont aujourd'hui nombreuses et variées :

- Organisation de l'emploi du temps d'une université, les ressources étant alors les enseignants, les élèves et les salles de cours, et les tâches étant les enseignements.
- Gestion du service des infirmières dans un hôpital.
- Gestion de la production d'une entreprise, les ressources seront les machines et le personnel, les tâches étant la production de biens.

Aujourd'hui, il existe quelques environnements d'édition de workflow : ActionWork, MicrosoftProject .

L'édition de worklfow dans ces environnements peut se décrire en trois étapes :

- Spécification de l'enchaînement temporel des différentes tâches.
- Allocation de ressources aux différentes tâches.
- Calcul de l'enchaînement des tâches de manière à prendre en compte la spécification de l'auteur et les contraintes sur les ressources. Par exemple, une ressource peut être partagée par deux tâches indépendantes temporellement, a priori, et de ce fait, ces deux tâches ne pourront s'exécuter en parallèle.

Au cours de l'expérience réalisée par Michael Tissot pendant son stage de maîtrise sous ma direction, nous nous sommes intéressés essentiellement à la spécification de l'enchaînement temporel des différentes tâches [Tissot00]. Aspect qui nous a semblé le plus proche de l'édition de documents multimédias, et qui n'était pas résolu de manière satisfaisante dans ces outils.

Nous allons, dans un premier temps, formaliser la définition d'un workflow (section 3.1), dans un deuxième temps nous dégagerons les principaux problèmes liés à l'édition de workflow (section 3.2) ceci afin de dégager les points communs avec les services offerts par Kaomi. Enfin dans la dernière partie de cette section (section 3.3) nous présenterons l'implémentation de cet éditeur ainsi que les apports et les limites de l'utilisation d'une boîte à outils telle que Kaomi dans ce contexte.

3.1 Modélisation d'un workflow

Un workflow peut se décomposer en tâches élémentaires et en tâches composites :

- Une tâche élémentaire peut être définie par :
  - Un intervalle pour définir l'instant de début de la tâche, par exemple la tâche doit commencer entre le 5 et le 10 juin.
  - Un intervalle pour définir l'instant de fin de la tâche, par exemple la tâche doit finir entre le 22 et le 24 juin.
  - Un intervalle de durée : la durée de la tâche est comprise entre 5 et 10 jours.
  - La contrôlabilité de la tâche, une tâche est contrôlable si avant de la commencer, on peut fixer sa durée, par exemple, la réalisation de la tâche T1 durera 6 jours, dans le cas contraire on dit que la tâche est incontrôlable. C'est le cas, par exemple, si sa réalisation dépend de conditions externes que l'on ne peut pas contrôler.
  - Un ensemble de ressources nécessaires à la réalisation de la tâche.
  - Un ensemble de ressources allouées à la réalisation de la tâche.
- Une tâche composite est définie par :
  - Un ensemble de tâches élémentaires ou composites.
  - Un ensemble de relations entre ces tâches : les tâches commencent en même temps, une tâche doit impérativement être finie à telle date, deux tâches doivent se dérouler en séquence.
  - Un intervalle pour définir l'instant de début de la tâche, par exemple la tâche doit commencer entre le 5 et le 10 juin.
  - Un intervalle pour définir l'instant de fin de la tâche, par exemple la tâche doit finir entre le 22 et le 24 juin.
  - Un intervalle de durée : la durée de la tâche est comprise entre 5 et 10 jours.
  - La contrôlabilité de la tâche, une tâche composite est contrôlable si toutes les tâches élémentaires qui la composent sont contrôlables.
  - Un ensemble de ressources nécessaires à la réalisation de la tâche.
  - Un ensemble de ressources allouées à cette tâche.

On peut remarquer que la définition d'un workflow ressemble à la définition d'un document multimédia, avec une priorité plus importante sur l'aspect gestion de ressources. Cependant, on note que l'on peut assimiler les besoins liés au partage de ressource aux aspects de qualité de services. Aspect que l'on retrouve dans le cadre des documents multimédias avec les besoins en débit du réseau, mémoire et CPU de la machine.

3.2 L'édition de workflow

Des différents systèmes auteur de workflow que nous avons étudiés [Tissot00], nous pouvons dégager un ensemble de caractéristiques communes :

- La représentation des informations temporelles :
  - Soit à base de vue temporelle proche d'un paradigme de temps absolu (Microsoft Project), c'est-à-dire que les objets sont visualisés sur un axe temporel, et leur longueur est proportionnelle à leur durée.
  - Soit à base d'une visualisation d'un graphe de dépendance entre les tâches (ActionWork).
- Les outils auteur de workflow se limitent souvent à la spécification du workflow et à sa résolution. Dans les outils étudiés, il n'y a aucune aide au diagnostique de la cohérence du workflow, les outils se contentent de dire, à la fin de la spécification que le workflow est cohérent ou non et de produire une solution dans le cas d'une spécification cohérente. De la même façon, il n'y a aucun moyen de paramétrer la résolution du workflow, de manière à, par exemple, minimiser la durée totale du workflow ou pour maximiser l'utilisation de certaines ressources.
- Les outils auteur de workflow offrent une édition directe très pauvre, c'est-à-dire que lorsque l'auteur manipule la représentation graphique du workflow et les contraintes entre les tâches ne sont pas maintenues graphiquement.

C'est pour essayer d'améliorer les différents points ci-dessus que nous avons construit un environnement de workflow au-dessus de Kaomi pour tirer parti des services d'édition/visualisation de la vue temporelle, et des services de vérification de cohérence incrémentaux. En effet, la présentation ci-dessus montre que la spécification temporelle de l'enchaînement des tâches pour un éditeur de workflow est très proche de l'édition d'un document multimédia. Les différences se situent au niveau de l'allocation de ressources, ainsi que dans la datation absolue de certains événements.

3.4 Implémentation de Workflow Editeur

La première étape a été de définir une DTD XML pour modéliser un workflow. A partir de cette étape, il a ensuite fallu définir une structure de données permettant de modéliser sous forme de *Documents* (format pivot de Kaomi) un workflow. Pour cela, nous avons dans un premier temps défini pour chaque tâche un média texte qui contenait le nom de la tâche et la liste des ressources nécessaires, ensuite nous lui avons défini des propriétés spatiales arbitraires, et enfin nous avons défini les propriétés temporelles entre objets en suivant la spécification du workflow. L'implémentation de cette tâche utilise la classe ElementMultimédia qui offrait tous les mécanismes pour stocker et éditer les informations nécessaires.

Au cours de la traduction de la spécification d'un workflow vers un ElementMultimédia, nous avons été limité par l'expressivité de notre modèle temporel, nous n'avons par exemple pas la possibilité de spécifier des dates absolues. En effet, dans Kaomi, nous utilisons des dates relatives au début du document, et nous n'avons pas le moyen, sans introduire un facteur d'échelle trop important pour les résolveurs, d'avoir un système de date absolue. De ce fait, la phase de vérification de cohérence et de formatage à chaque opération d'édition prendrait un temps trop important pour l'auteur.

Cette limitation n'est pas spécifique au domaine du workflow, dans des documents conçus pour des domaines d'application comme l'enseignement, on retrouve le même besoin. Par exemple, les étudiants ne peuvent accéder à un cours ou une correction d'exercice qu'à partir d'une date précise.

Une fois la définition d'une classe tâche réalisée, il a fallu définir le fichier de ressources nécessaires à l'édition de tâches et notamment à l'édition des ressources. Dans la palette résultante, l'auteur peut choisir les ressources utilisées et leur taux d'utilisation (Figure VI-12).

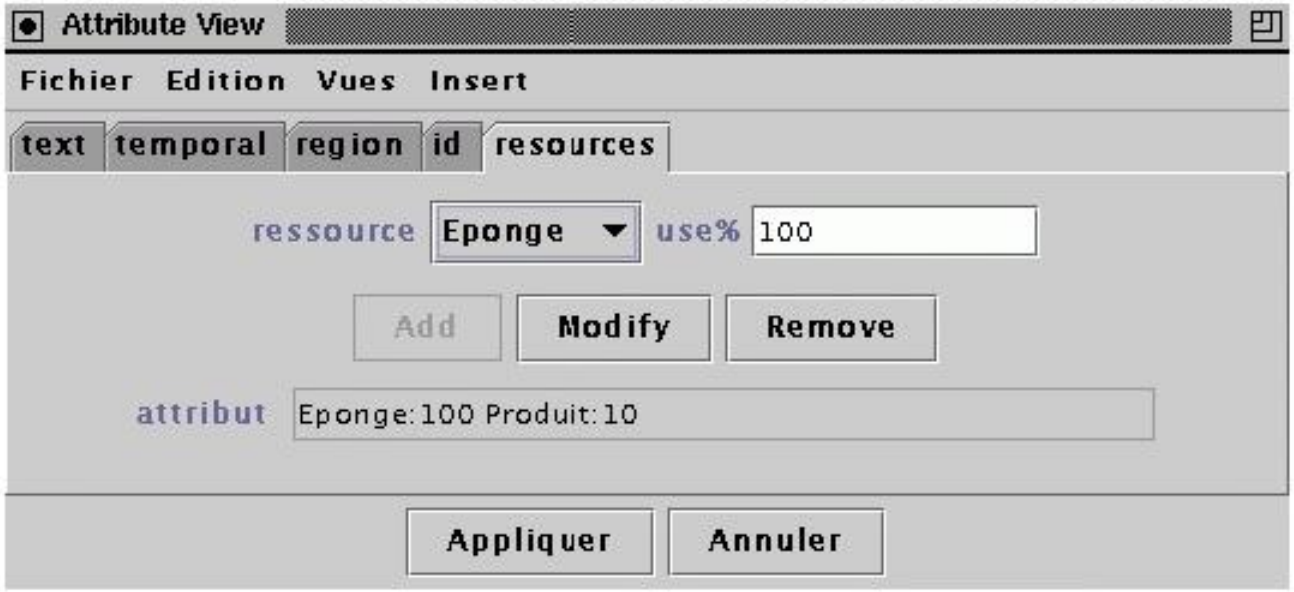


Figure VI-12 : Palette d'attributs de Workflow Editeur

Une fois la traduction du workflow dans une structure de données compatible avec la boîte à outil effectuée, nous avons pu utiliser tous les services de celle-ci : visualisation de l'enchaînement temporel, vérification de la cohérence, aide au formatage pour des tâches déterministes uniquement.

Dans l'exemple de la Figure VI-13, nous pouvons voir le résultat du placement temporel d'un workflow. Ce workflow organise la journée de Michael qui s'installe dans son nouvel appartement. Son après-midi est organisé en deux grandes parties : le repas et le nettoyage de l'appartement. Pour la réalisation de la deuxième tâche, Michael sera aidé d'un camarade.

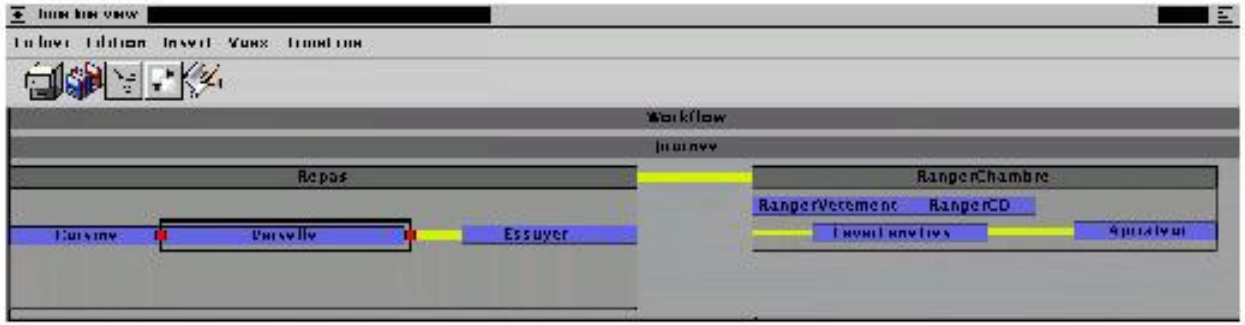


Figure VI-13 : Vue temporelle de Workflow Editeur

Par rapport à l'utilisation des résolveurs de contraintes, une première expérience a été menée pour intégrer la prise en compte des ressources lors du formatage. L'idée de cette expérience était d'intégrer les dépendances liées aux ressources, ainsi que le calcul de l'allocation de ressources dans le système de formatage. Les premiers travaux ont donné des résultats encourageants car ils permettent d'avoir un formatage qui prend en compte l'utilisation de

ressources, cependant ils n'ont pu être menés à terme à l'heure actuelle.

3.4 Bilan de Workflow Editeur

Les principaux apports de l'utilisation d'une boîte à outils comme Kaomi dans l'édition de workflow sont :

- Aide à la visualisation des informations temporelles.
- Edition directe par manipulation avec maintien des relations temporelles lors de l'édition, ce qui permet à l'auteur de mieux identifier la source de l'erreur.
- Vérification des incohérences et détection incrémentale des erreurs.
- Aide au formatage du workflow.
- Simulation de la dimension temporelle du workflow.

Parmi les limitations de la boîte à outils rencontrées, on peut citer la nécessité d'intégrer un formatage temporel avec gestion des ressources physiques ainsi que la nécessité d'intégrer des dates absolues. Un autre problème est la disparité dans les échelles de temps.

4 Les bases de données documentaires

Dans le cadre d'une coopération avec l'Aérospatiale ([Aérospatiale99]) nous avons utilisé Kaomi pour réaliser un prototype permettant d'éditer et de gérer un fond documentaire [Duluc00, Duluc00b]. Ce prototype a été réalisé par Frank Duluc dans le cadre d'une thèse Cifre en coopération avec les membres de l'équipe Opéra. Ce prototype permet de couvrir les différentes phases de la chaîne documentaire : l'édition, la gestion du fond, la production et la présentation des documents multimédias. Ce travail a été publié collectivement par Franc Duluc et les personnes d'Opéra [Duluc00].

L'idée était, dans un premier temps, de manipuler dans la base de documents un ensemble de fragments de documents multimédias, appelés *granules*, c'est-à-dire que l'on ne stocke pas seulement des médias dans cette base mais aussi des parties de documents. L'intérêt est de stocker un ensemble de médias agencés temporellement et spatialement. Dans un deuxième temps, à partir de ces fragments, l'auteur (et à plus long terme le système) génère une présentation qui doit être visualisée. L'apport d'une boîte à outils dans un tel processus est de fournir un mécanisme de création pour les différents fragments de la base documentaire, et, dans un deuxième temps, de permettre une édition et une visualisation des différents fragments pour en faire des documents à part entière. Ce prototype a été réalisé en utilisant les fonctionnalités Java pour faire l'interface avec les bases de données.

Ce prototype a permis de mettre en évidence la capacité d'utiliser une technologie multimédia dans les circuits de production documentaire. Cependant, il est nécessaire d'utiliser des processus automatiques de génération à partir des fragments de base comme pour les documents textuels. Pour réaliser cela, il est nécessaire d'utiliser des modèles de documents génériques qui prennent en compte le modèle temporel.

5 Evaluation quantitative de Kaomi et des environnements auteur

Comme j'ai pu le dire au début du chapitre IV, Kaomi est le résultat d'un travail que j'ai initié et qui a évolué grâce à la participation de nombreuses personnes. Afin de clarifier les contributions de chacun j'ai reporté, dans la Figure VI-14, une évaluation du nombre de lignes de code de Kaomi, des

différents environnements auteur construits à l'aide de Kaomi ainsi que de la contribution de chacun. On peut voir que le nombre de lignes de code spécifique est mineur par rapport à la boîte à outils et ce code consiste essentiellement à la définition d'un analyseur lexical spécifique. De plus, dans cette figure, j'ai reporté ma participation directe et indirecte (via l'encadrement de stagiaires) à cette boîte à outils et aux différents environnements auteur.

	Kaomi	MadeusEd.	SmilEd.	MHMLEd.	WorkflowEd.
Nombre de lignes de code	110000	5000	6000	3000	4000
% développé personnellement	20%	20%	35%	35%	30%
% développé par des stagiaires sous ma responsabilité	40%	30%	60%	0%	70%
Autres : doctorants, ingénieur	40%	50%	5%	65%	0%

Figure VI-14 : Code de Kaomi et des environnements auteur

Au cours de ces différentes expériences, la boîte à outils s'est enrichie et a été rendue de plus en plus extensible et générique. Cette extensibilité et cette généricité constituent sans aucun doute un point fort qui laisse présager de bonnes possibilités d'évolution dans le futur. Nous reviendrons sur ce point dans la conclusion.

6. Bilan des expérimentations

Nous venons de voir, au cours de ce chapitre, la présentation des expérimentations menées avec la boîte à outils Kaomi. Je vais en faire un bilan en trois points :

- *Modèle d'édition proposé.* Nous avons vu dans les chapitres III et IV qu'un environnement auteur devait, pour faciliter la tâche de l'auteur, lui fournir un formalisme d'édition de plus haut niveau que celui du format de présentation, tout en lui permettant d'exprimer tout ce que le format de présentation lui permettait. Au cours des expériences avec SMIL-Editeur et MHML-Editeur, nous avons vu comment se passe l'édition en combinant ces deux formalismes. Nous avons vu de plus, que les services de l'environnement auteur étaient dépendants du formalisme utilisé par l'auteur. Les services d'aide sont optimaux pour les parties spécifiées (ou compatibles) avec le formalisme de l'environnement auteur, et ils offrent un service minimal dans le cadre d'une édition spécifique. La boîte à outils permet au développeur de l'environnement auteur de spécialiser ces services pour le

langage utilisé.

- *Proposition d'environnement ayant plusieurs vues synchronisées*. Dans le chapitre III nous avons proposé un environnement permettant à l'auteur d'éditer son document dans plusieurs vues, utilisant pour chaque opération la plus adaptée à son besoin. Les différentes expérimentations menées ont permis de vérifier que ce concept de multivues est parfaitement adapté aux différents formalismes de spécification de documents multimédias et que le mécanisme d'extension des vues est un moyen d'adaptation aux formats de présentation (MHML).
- *Bilan de l'utilisation de Kaomi comme base de conception de systèmes auteur*. Trois systèmes auteur de documents multimédias ont été réalisés au-dessus de Kaomi. Nous avons vu dans la section précédente que le code spécifique nécessaire pour chacun de ces environnements auteur était mineur par rapport au code de la boîte à outils. Cela montre la pertinence de l'utilisation de Kaomi dans ce contexte.

Le bilan des expérimentations faites avec Kaomi ne nous permet de relever que très peu de points négatifs. Cela est principalement lié au fait que les manques rencontrés lors de la création des différents environnements auteur ont été directement implémentés dans la boîte à outils. C'est le cas par exemple du module de simulation réalisé initialement pour MHML-Editeur. La validation de cet environnement est plus poussée que les autres du fait que nous avons eu un retour de la part d'utilisateurs.

On peut noter cependant l'impossibilité de spécifier des dates absolues, ainsi que la réalisation de services de diagnostique des incohérences qui est quasi-inexistant.

7. Travaux futurs

Malgré les différents aspects positifs, Kaomi reste une boîte à outils qui doit s'étendre et s'enrichir. Les différentes extensions que l'on peut envisager sont classées ci-dessous en cinq catégories :

Médias : les deux premières extensions peuvent être considérées comme un travail d'implémentation et ne poseront aucun problème d'intégration dans la boîte à outils. Les deux dernières extensions nécessiteront par contre un travail de réflexion plus important pour les intégrer complètement au processus d'édition.

- *Intégrer l'édition des médias* : il est contraignant pour un auteur d'utiliser des outils différents pour la création des différents types de médias. L'environnement auteur doit fournir un ensemble de primitives élémentaires pour éditer les différents médias et leur affecter des transformations ou des effets de style élémentaires (Director, PowerPoint).
- *Intégrer une édition plus fine des médias* en identifiant des sous-parties à l'intérieur de ceux-ci, par exemple, par la définition de scènes dans une vidéo. Cela permettra une synchronisation de ces éléments avec d'autres médias du document. La principale difficulté, outre la définition de sous-parties dans les médias est le facteur d'échelle que cela introduit dans la boîte à outils. Un travail sur l'édition de vidéos structurées est actuellement réalisé par Tien Tran Thuong, étudiant en thèse. L'objectif de ce travail est de permettre à l'auteur de définir une structure pour une vidéo et de l'utiliser pour synchroniser des scènes de la vidéo avec des médias du document [Roisin00]. Ce travail se concrétise actuellement par l'introduction d'une vue vidéo structurée dans Kaomi.
- *Accéder à des bases de données*: le couplage d'un environnement auteur avec une base de données pour accéder aux médias permettra dans un premier temps d'étendre les capacités de recherche et d'insertion des médias lors de l'édition. Dans un deuxième temps, les documents pourront contenir des requêtes vers des bases de données intégrant ainsi dynamiquement ces médias tout en maintenant un ensemble de propriétés spatiales et temporelles au document. On retrouve là la notion de granule de Franck Duluc[Duluc00b]. Cependant, cela posera un problème de formatage

dynamique du document pour prendre en compte cette intégration de manière optimale.

- *Implémenter un ensemble plus large de médiateurs* : les bibliothèques libres utilisées permettent de jouer l'essentiel des standards de médias. Cependant, dans le cadre d'un environnement auteur ayant un objectif de diffusion large, il est nécessaire de permettre l'ouverture vers de nouveaux standards de médias (SVG) ainsi que l'intégration d'un ensemble de médias dans des formats propriétaires (Word, Excel, Director). Les difficultés de cette intégration sont liées à la granularité de l'insertion de tels médias et donc à la synchronisation fine d'une partie de ces médias avec d'autres éléments du document.
- *Intégrer des comportements indéterministes :* l'intégration de médias indéterministes nécessite une adaptation de la notion de formatage et de cohérence [Layaïda97]. Vu la complexité théorique du traitement de l'indéterminisme, une première approche (traitant partiellement le problème) serait d'introduire une notion de cohérence locale. Une première expérience d'édition /présentation a été menée dans le cadre de MHML-Editeur.

Hypertexte:

- *Intégration du processus de lecture dans le document*. Cette extension permettra de ne plus considérer un document comme une entité isolée, mais comme faisant partie d'un tout comme, par exemple, d'un processus d'apprentissage. On retrouve là les idées rencontrées dans le workflow. L'environnement de lecture doit être capable de gérer le flux de lecture, c'est-à-dire l'enchaînement de parcours entre les différents documents faisant partie de ce processus. Cela peut se réaliser par l'utilisation de liens conditionnels.
- *Visualisation*: on peut diviser l'extension des services de visualisation de la structure hypertexte en deux étapes :
  - *Intégrer un service minimum de visualisation de la structure hypertexte* : il est nécessaire de permettre à l'auteur de visualiser cette structure et si possible, de naviguer facilement à l'intérieur de celle-ci, en offrant par exemple une visualisation graphique des liens hypertexte du document.
  - *Intégrer un service évolué de visualisation et d'édition de la structure hypertexte*. Les principales difficultés seront de permettre à l'auteur de percevoir cette structure pour de gros documents. Les représentations actuelles des structures hypertexte de document sont limitées par le facteur d'échelle (voir Chapitre II).

Document: au cours de cette thèse nous avons considéré les médias comme des éléments de base, nous avons restreint le pouvoir d'expression de l'auteur pour faciliter l'édition. Il est nécessaire d'étendre ces deux limites pour offrir un environnement auteur complet.

Permettre l'accès à un langage de programmation : il existe toujours des situations où, quel que soit l'environnement auteur utilisé, certains auteurs spécialisés auront besoin d'accéder ponctuellement à la puissance d'un langage de programmation (lancement de programme externe, tests sur la configuration de l'environnement de lecture du document). Il est donc nécessaire de fournir de tels points d'entrée. Cette intégration nécessitera la vérification de l'intégrité d'une telle édition par rapport au reste de la spécification du document.

Modèle de document / feuille de style : la notion de modèle de document utilisée dans l'édition textuelle n'est pas encore intégrée à l'édition de document multimédia. Cette intégration soulève deux problèmes :

- *Intégrer la notion de modèle de documents* aux documents multimédias en s'inspirant des travaux réalisés pour les documents textuels ([Quint87], [Furuta88]). Cela permettra de définir des classes de documents et de faciliter l'édition du document par l'utilisation de feuilles de style, en dégageant ainsi l'auteur de la définition du positionnement spatial et temporel de ces objets.
- *Définir des modèles de documents* :l'intégration de modèles de documents permettra de faciliter l'édition, cependant l'écriture de ces modèles de documents ne se fait pas encore de manière conviviale. Il faudra fournir des moyens interactifs pour définir de tels modèles que ce soit pour les sources de données ou pour les feuilles de styles. Un premier travail dans ce sens est en cours au dessus de Kaomi : il est réalisé par Lionel Villard,

étudiant en thèse [Villard00] et vise principalement a intégrer les transformations des données incrémentales pour l'intégrer au processus d'édition.

Souplesse de présentation: nous avons vu que la définition relative du document permet de simplifier la tâche d'édition. Cette souplesse définie dans le document peut être aussi utilisée lors de la présentation du document.

- *Formatage dynamique (en cours de présentation) du document :*l'intégration dynamique de médias ou l'adaptation du formatage temporel et spatial à l'évolution des conditions de présentation nécessitent la mise en place de formateurs prenant en compte cette composante.
- *Intégration d'un service de gestion de la qualité de service* pour la vue de présentation : la manipulation de médias dynamique nécessitant de nombreuses ressources physiques (mémoire, bande passante), une politique de pré-chargement et d'accès aux médias efficace est nécessaire. L'utilisation d'un graphe temporel dans la vue de présentation permet de connaître statiquement le cours de l'exécution et permet ainsi au système, à un instant t, d'anticiper le chargement des médias qui seront utilisés dans le futur. Un travail est mené par Maximilien Laforge pour améliorer le service proposé par la vue de présentation [Laforge00]. La difficulté sera de généraliser ce travail aux documents imprédictifs.
- *Alternative sur les médias :*une définition relative du document devrait faciliter entre autre le changement des médias dynamiquement. De ce fait, l'écriture de documents multilingues pourrait se réaliser en associant aux objets multimédias de Kaomi un ensemble de médias (propre à chacune des langues), et le système pourrait ainsi, au moment de la présentation choisir l'ensemble de médias adapté tout en conservant la spécification du comportement temporel et spatial entre les objets multimédias.

Fonctions utilisateur avancées :enfin, il est aussi nécessaire d'étendre certaines fonctions d'édition pour notamment prendre en compte l'édition coopérative de documents.

- *Gestion d'utilisateurs sur le document et de leurs droits*: les documents peuvent s'adapter en fonction des lecteurs, cette adaptation dépend en partie des droits que le lecteur possède au moment de l'accès au document. Par exemple, dans le cas d'un document représentant un cours, on peut imaginer qu'un étudiant n'ait pas le droit de regarder la correction d'un exercice avant une certaine date.
- *Gestion de l'édition coopérative de documents multimédias :*de nombreux travaux portent sur l'édition coopérative de documents textuels ou hypertexte ([Decouchant96], [Byzance00], [Séraphine01]). L'édition de documents multimédias met en oeuvre de nombreuses personnes travaillant à différentes étapes du processus de conception. Il est donc naturel de souhaiter un environnement qui leur permette de collaborer efficacement entre elles.

8. Conclusion

Dans ce chapitre nous avons d'une part décrit les différentes réalisations effectuées avec Kaomi, et d'autre part, à partir d'un bilan de ces expérimentations, porté à la fois sur le formalisme d'édition proposé et sur l'utilisation de la boîte à outils, nous avons dégagé un ensemble de pistes pour étendre les capacité d'une telle boîte à outils.

Conclusion

1. Rappel des objectifs et contexte

Le projet Opéra étudie les problèmes posés par la conception, l'édition et la présentation des documents multimédias. Mon travail de thèse effectué dans cette équipe, a concerné la conception et la réalisation d'un système d'édition qui permet de prendre en compte, pendant le processus d'édition, les besoins des auteurs, novices ou experts.

La construction d'un environnement d'édition est soumise à des contraintes multiples. Ces contraintes sont liées d'une part à la multitude de formats de spécification de documents multimédias, et d'autre part aux différents objectifs et aux compétences des auteurs.

La plupart des systèmes d'édition actuels proposent des méthodes d'édition très liées au format de document sous-jacent. De ce fait, l'édition ne propose pas, ou peu, de fonctionnalités pour aider l'auteur et automatiser certaines tâches.

2. Démarche de travail et bilan

Mon travail de thèse s'est organisé en deux grandes parties, ce découpage se retrouve dans la thèse :

Environnements auteur de documents multimédias

La première partie est axée autour des environnements auteur de documents multimédias. J'ai tout d'abord analysé la situation courante des différents environnements auteur. Au cours de cette analyse, j'ai dégagé quatre principaux formalismes d'édition de documents multimédias, et j'ai montré comment les environnements auteur permettaient de les éditer.

Le premier résultat de la thèse consiste, à partir de cette étude, à la définition de ce que serait un environnement d'édition permettant à un auteur de spécifier à la fois simplement et de manière complète son document multimédia.

Le deuxième résultat présenté est la boîte à outils Kaomi. Elle doit permettre de construire de tels environnements auteur, tout en offrant aux concepteurs de ces environnements une mise en commun des services.

Systèmes de contraintes dans l'édition de documents multimédias :

A partir de ces résultats et des différentes expériences menées, je me suis plus particulièrement intéressé plus particulièrement aux contraintes et aux systèmes de résolution de contraintes. L'étude des résolveurs de contraintes constitue la deuxième partie de la thèse. Dans une première étape, j'ai dégagé un ensemble de besoins qui n'étaient à ce jour pas satisfaits. J'ai ensuite démontré que les technologies basées sur les contraintes pouvaient apporter des réponses intéressantes aux besoins liés à l'édition de documents multimédias. Pour cela, j'ai étudié les différents mécanismes de résolution

ainsi que leur utilisation optimale dans le contexte de l'édition de documents multimédias.

Ainsi, le troisième résultat de la thèse consiste en l'évaluation des différents types de résolveurs dans notre contexte. Différentes expériences m'ont permis d'obtenir des résultats qualitatifs et quantitatifs sur les différentes approches de résolution de contraintes dans notre contexte d'édition de documents multimédias. À partir de cette analyse, j'ai proposé un algorithme hybride offrant un bon compromis entre les performances et la qualité des différentes réponses calculées.

Le quatrième résultat de la thèse est l'implémentation d'environnements auteur basés sur la boîte à outils Kaomi. Ces environnements auteur sont représentatifs des différents formalismes d'édition et de spécification de documents multimédias mais aussi de contextes applicatifs différents comme l'édition de workflow ou la gestion de bases documentaires. Ces expériences ont permis de valider d'une part la généricité de la boîte à outils, et d'autre part l'apport des contraintes au travers des services d'édition avancés qui ont été mis en oeuvre.

Enfin, à partir de ces expériences, j'ai dégagé les différentes utilisations futures des techniques à base de contraintes pour le contexte de l'édition de documents multimédias.

3. Perspectives

J'ai choisi de classer les perspectives de ce travail selon trois axes. Le premier axe concerne les aux apports possibles de l'utilisation des contraintes dans un contexte d'édition de documents multimédias de manière à exploiter pleinement le travail réalisé dans cette thèse. Dans un deuxième temps, je présenterai la situation de la boîte à outils Kaomi vis-à-vis des évolutions des documents multimédias. Et enfin, dans un troisième temps, je présenterai les différents points liés aux contraintes, qui doivent être améliorés pour rendre cette technologie complètement exploitable dans un contexte applicatif.

3.1 Perspectives sur l'utilisation des technologies contraintes

Dans le cadre de l'édition de documents multimédias, les perspectives d'utilisation des technologies contraintes sont nombreuses. Dans un premier temps, je vais présenter les différentes utilisations possibles de ces techniques qui sont des applications directes des travaux réalisés, je présenterai ensuite une problématique plus générale sur l'adaptation de documents qui se développe aujourd'hui du fait de l'essor de la téléphonie mobile et des ordinateurs de poche.

- **Prototypage rapide de documents**

Un des besoins essentiels lors de la conception de documents est la possibilité de prototyper rapidement une version de manière à pouvoir avoir un aperçu du document [Bollard99]. Dans ce contexte, un formalisme de haut niveau à base de contraintes permet de spécifier de manière partielle un document. L'auteur, en peu de temps, peut obtenir une maquette qui lui permet d'avoir une idée précise de ce que sera son document dans sa forme finale et même de naviguer dans un espace de solutions. Cette phase de prototypage est très utile lors de la production de documents pour, par exemple, avoir la possibilité de proposer des maquettes différentes aux autres auteurs, aux collaborateurs ou même aux clients. Lesquelles serviront ensuite de point de départ pour les discussions entre les concepteurs.

- **Effets de style ou macros adaptables**

Dans le cadre des documents multimédias, il arrive que l'on ait besoin de définir un effet de présentation pour un ensemble de *n* objets. Que se passe-t-il, lorsque l'on constate qu'il faut insérer un objet de plus dans cet effet ? Par exemple, l'auteur définit une animation de cinq objets qui s'animent

pour se regrouper et former un titre. Si l'auteur désire insérer un sixième objet dans son animation, il est souhaitable qu'il n'ait pas à recalculer le placement et le mouvement de tous ces objets. Les techniques à base de contraintes peuvent être utiles pour calculer une nouvelle solution. En effet, cela permet au programmeur de l'effet de style de ne pas écrire toutes les solutions possibles dont un utilisateur aura besoin. Il décrira seulement les contraintes à respecter entre les objets et laissera le calcul du placement exact des objets au résolveur de contraintes.

- **Adaptation de document et contraintes**

Un des intérêts d'utiliser des techniques à base de contraintes est de permettre le formatage, soit statiquement au moment de l'édition du document, soit au moment de la présentation. Ce dernier cas permet d'adapter le document au lecteur ou aux conditions de présentation.

Je présente ici les principales utilisations de ces possibilités de formatage.

- **Adaptation de documents générés**

Un des intérêts de l'utilisation de schémas prédéfinis (chapitre II section 4.1) est la simplicité de spécification pour l'auteur. Un des intérêts d'une technologie contraintes combinée avec les schémas prédéfinis est de rendre plus flexible ces schémas. Imaginons par exemple un schéma pour présenter en séquence une liste de photographies. Cette liste peut être le résultat d'une requête vers une BD, ou une liste de photographies fournies par l'auteur. L'utilisation de résolveurs et d'un formalisme de spécification à base de relations permet au lecteur de fixer la durée globale de la présentation et le système déduira automatiquement la durée de présentation pour chacune des images de la séquence en fonction de leur nombre.

- **Multi-langage**s

Une des applications intéressantes qui tire parti d'un formatage souple est de pouvoir réaliser des documents multilingues, formatés en fonction de la langue. Par exemple, un média audio n'aura pas la même durée en anglais ou en français. De la même façon, un texte anglais n'aura pas la même longueur que sa traduction française. De ce fait, avoir un document contenant de la flexibilité et un outil permettant de formater ce document est un apport intéressant pour l'auteur, qui laisse ainsi le soin au formateur de calculer au mieux le placement des objets du document, en concervant si possible la durée intrinsèque de ces médias. L'auteur ne spécifie ainsi qu'un document. Le système le formatera alors de façon optimale pour chacune des langues.

- **Multi-lecteur**

On peut, de la même manière, adapter un document en fonction des capacités du lecteur. Par exemple, dans le cadre de l'enseignement, on peut concevoir des présentations différentes d'un même cours en fonction de l'élève et/ou de son niveau d'apprentissage. De la même manière, dans le domaine médical, le dossier d'un patient doit être présenté de manière différente à un spécialiste, à l'infirmière traitante ou au patient lui-même. La visualisation ou non de certains objets va donc entraîner une phase d'adaptation du document qui pourra être prise en charge par le résolveur de contraintes.

- **Formatage dynamique**

Une autre forme d'adaptation qui peut être traitée par les contraintes est l'adaptation dynamique du document au cours de la présentation. Cette adaptation se fait par rapport aux conditions du réseau, de la charge du système, de la machine (taille de l'écran,). À partir d'un document formaté partiellement statiquement, le système de présentation va appliquer une (ou plusieurs) phases de formatage dynamique pour adapter la présentation aux nouvelles conditions.

3.2 L'avenir du multimédia : de nombreux formats

Nous avons vu qu'il existait de nombreux formats de spécification pour les documents multimédias, il faut aujourd'hui prendre en compte ce paramètre et essayer de fournir des outils auteur capables de manipuler un ensemble de formats. L'émergence de standards comme XML permet d'homogénéiser la manière de décrire les documents et de spécifier les propriétés sous forme d'attributs. Ainsi, les langages SMIL et Madeus sont décrits en XML. L'intérêt d'avoir un moyen standardisé de décrire les données permet de concevoir des outils communs pour décrire la présentation.

Néanmoins, l'émergence de standards comme SMIL 2.0 nécessitera une adaptation des outils auteurs.
SMIL 2.0

Les objectifs de SMIL 2.0 sont, dans un premier temps, d'étendre l'expressivité de SMIL 1.0 et, dans un deuxième temps, de rendre plus modulaire la spécification de SMIL.

Les principales extensions de l'expressivité ont été réalisées dans la définition de la synchronisation temporelle. L'auteur peut maintenant, par exemple, définir des objets maîtres dans les relations de synchronisation. Il peut aussi spécifier de manière plus complète l'interactivité dans le document.

Le deuxième changement important dans SMIL est la spécification de modules. Les principaux modules sont : structure, lien, média, animation, synchronisation,

Ces différents modules permettent de définir la présentation (spatiale, temporelle) ainsi que la structure de navigation d'un document. Un des intérêts est aussi de permettre l'utilisation d'un sous-ensemble de modules dans d'autres contextes. On peut noter que de ce fait, SMIL 2.0 permet de définir un ensemble de langages à partir de sous-ensembles de modules.

On peut noter que ces deux extensions vont dans le sens des travaux menés dans Kaomi. L'augmentation de l'expressivité dans le module temporel fait que SMIL se rapproche de langages comme MHML. Ainsi, les différentes facilités d'édition fournies par le système auteur restent pertinentes.

On se rapproche donc de documents dont le contenu est décrit dans une syntaxe XML, et dont la présentation est définie grâce à un ensemble de modules SMIL. Ce sous-ensemble évolue en fonction du contexte. Cela nous incite à penser que, dans certains cas, on voudra spécialiser le système auteur pour ce sous-ensemble de modules. L'architecture de la boîte à outils est adaptée à ce besoin.

3.3 Extensions à apporter aux technologies contraintes

L'utilisation de contraintes dans un environnement d'édition permettra à l'utilisateur de paramétrer la résolution. Pour cela il est nécessaire d'étendre l'environnement auteur actuel sur deux points :

- **Politique de formatage adaptable :**comme nous avons pu le voir précédemment, un des points positifs des résolveurs est qu'ils permettent de définir une politique de formatage quel que soit le langage édité. Cependant, cette politique de formatage est dépendante du contexte d'utilisation. Dans certains cas, l'auteur désirera que tous ses objets aient la même durée (présentation de photographies en séquence), et dans d'autres cas, il désirera fixer une durée pour certains objets et laisser le formateur faire au mieux pour le reste, de manière, par exemple, à limiter la bande passante moyenne nécessaire pour visualiser le document. On doit donc fournir à l'auteur le moyen de définir une politique de formatage, ce qui n'est pas simple à faire du fait que le bon paramétrage des poids associés aux contraintes se fait souvent de manière empirique. Il est donc nécessaire dans un premier temps de voir comment les

résolveurs de contraintes peuvent faciliter ce contrôle sur le mécanisme de résolution et dans un deuxième temps d'étudier la façon dont la phase de paramétrisation pourrait être assistée par le système auteur.

- **L'aide au diagnostic** : aujourd'hui, peu de travaux existent sur l'explication de la solution calculée par un résolveur. Cette aide est essentielle si on veut que l'utilisateur puisse paramétrer et configurer le formatage, pour que ce dernier satisfasse au mieux ses besoins. De même, il reste des travaux à effectuer dans le domaine de la compréhension d'un système de contraintes incohérent.

Nous avons vu que l'utilisation des contraintes s'est avérée concluante dans le cadre de documents déterministes. L'intégration de relations spatio-temporelles et de médias indéterministes vont soulever de nouveaux problèmes (facteur d'échelle, médias non formatables statiquement) et introduire de nouveaux besoins (formatage dynamique). Pour répondre à ces besoins de nombreux travaux théoriques visent à étendre l'expressivité des CSP.

- **Dynamique et fuzzy CSP** : des travaux formels ont été réalisés dans le cadre des contraintes pour prendre en compte des comportements imprédictifs. Par exemple, les fuzzy CSP [Verfaillie95] semblent offrir un formalisme intéressant. Dans ce formalisme, chaque contrainte est définie non pas par l'ensemble des valeurs qui la satisfont mais par un ensemble flou : l'ensemble des valeurs qui la satisfont plus ou moins. Dans ce cadre, une contrainte est satisfaite à un certain degré plutôt que satisfaite ou insatisfaite. La richesse de ce modèle permet de prendre en compte à la fois des préférences sur les valeurs et des priorités sur les contraintes. Dans ce modèle, l'acceptabilité globale d'une solution est graduelle. Les meilleures solutions sont celles qui maximisent le degré de satisfaction de la moins satisfaite des contraintes. C'est-à-dire, que si nous définissons le coefficient de qualité d'une solution comme la valeur du minimum des degrés de satisfaction des contraintes composant le problème, alors une bonne solution est celle qui a le coefficient de qualité le plus important. La complexité est du même ordre de grandeur que pour les CSP. Cette richesse permet d'exprimer les différentes préférences nécessaires à la résolution du système de contraintes lors de la phase d'édition.
- **Contraintes multidimensionnelles :**l'introduction de ces nouveaux besoins d'expression font augmenter sensiblement le nombre de contraintes. Actuellement nous avons un système de contraintes pour chacune des dimensions (temporelle, spatiale), nous pourrions en avoir un pour calculer, par exemple, les polices de caractères, la taille de ces polices, la couleur, Cependant, si on spécifie des dépendances entre ces différents éléments, les graphes de contraintes obtenus sont trop complexes pour les résolveurs actuels. Une théorie a été développée pour prendre en compte de tels problèmes [Verfaillie95], cependant peu de résolveurs prennent en compte ces particularités.

Références

[Adobe-Premiere00] Adobe, "Premiere, Manuel de référence", disponible à : http://www.adobe.com/

[Aérospatiale99] Aérospatiale, "Convention d'étude", BISM/SA 666.0012/99, 1999.

[Aggoun93] Aggoun A. et Baldiceanu N., "Extending CHIP in order to solve complex scheduling and placement problems", In Mathematical and Computer Modelling, vol. 17, n°7, pp. 57-73, 1993.

[AimTech96] Aim Tech, "Icon Author 6.0", Manuel de référence, 1996.

[Amaya00] Amaya, "Manuel de référence", disponible à : http://www.w3.org/Amaya/

[Apache-Xerces00] Apache, "Xerces", disponible à : http://xml.apache.org/xerces-j/index.html

[Asymetrix99] Asymetrix, "Toolbook II, manuel de référence", 1999.

[Badros98] Badros G. J. et Borning A., "The Cassowary Linear Arithmetic Constraint Solving Algorithm: Interface and Implementation", Technical Report UW-CSE-98-06-04, juin 1998.

[Badros00] Badros G. J., Nichols J. et Borning A., "Scwm--An Intelligent Constraint-Enabled Window Manage", AAAI Spring Symposium on Smart Graphics, mars 2000.

[Bailey98] Bailey B., Konstan J., Cooley R. et Dejong M., "Nsync - A Toolkit for Building Interactive Multimedia Presentations", Proceeding of the 6th ACM Multimédia Conference, Bristol,1998.

[Balter94] Balter R., Lacourte S. et Riveill M., "The Guide Language: Design and Experience ", The Computer Journal 37 , pp. 519-530, décembre 1994.

[Beldiceanu 94] Beldiceanu N. et Contejean E., "Introducing global constraint in CHIP", Journal of Mathematical and computer Modelling, 20(12), pp. 97-123, 1994.

[Bellynck99] Bellynck V., "Introduction d'une vue textuelle synchronisée avec la vue géométrique primaire dans le logiciel Cabri-II", Thèse de Doctorat, Université Joseph Fourier, 1999.

[Bes98] Bes F., "Spécification hiérarchique de scénarios temporels à base de contraintes", Rapport de D.E.A. informatique, Université Joseph Fourier , 1998.

[Bessière91] Bessière C., "Arc-consistency in Dynamic Constraint Satisfaction Problems", *Proceedings Of AAAI-91*, pp. 221-226, Anaheim, CA, 1991.

[Bessière94] Bessière C. et Cordier M-O., "Arc consistency and arc consistency again", *Artificial Intelligence*, 65, pp. 179-190, 1994.

[Bétrancourt98a] Bétrancourt M., Pellegrin A. et Tardif L. (1998). Using a Spatial Display to Represent the Temporal Structure of Multimedia Documents", Proceedings of the International Conference of the Cognitve Science Society of Ireland, Mind III 17 - 19, Dublin, août 1998.

[Bétrancourt98b] Bétrancourt M., Tardif L. et Pellegrin A., "Conception et évaluation d'un outil d'édition et présentation de documents multimédia", Colloque ERGO'IA 98, pp. 69-78, Biarritz, novembre 1998.

[Bétrancourt99] Bétrancourt M., "Arbre d'opérateurs vs. contraintes : comparaison expérimentale de deux approches de spécification temporelle", Rapport d'étude pour Alcatel, INRIA, 1999.

[Bétrancourt00] Bétrancourt M., Pellegrin A. et Tardif L. "Using a Spatial Display to Represent the Temporal Structure of Multimedia Documents", S.O'Nuallain, (Ed.) Spatial Cognition, John Benjamin : The netherlands, 2000.

[Beurivé00] Beurivé A., "Un logiciel de composition musicale combinant un modèle spectral, des structures hiérarchiques et des contraintes", présenté aux Journées d'Informatique Musicale JIM 2000, Bordeaux, 2000.

[Bonhomme98] Bonhomme S., "Transformations de documents structurés : une combinaison des approches déclaratives et automatiques", Doctorat d'informatique, Université Joseph Fourier, décembre 1998.

[Bonhomme99] Bonhomme S. et Roisin C., "Transformation de structures XML", Le Micro Bulletin Thématique : L'information scientifique et technique et l'outil Internet - Cnrs, num. 3, pp. 145-160, mai 1999.

[Bollard99] Bollard M., "Madeus, un environnement auteur de documents multimédia", DEA sciences cognitives, Université Pierre Mendès France, Juin 1999.

[Borning98] Borning A. et Freeman-Benson B., "Ultraviolet: A Constraint Satisfaction Algorithm for Interactive Graphics", Constraints, Special Issue on Constraints, Graphics, and Visualization, Vol. 3 No. 1, pp. 9-32, avril 1998.

[Borning87] Borning A., Duisberg R., Freeman-Benson B., Kramer A. et Woolf M . "Constraints hierarchies". In ACM Object-oriented Programming systems, Languages and Applications, pp 48-60, 1987.

[Borning92] Borning A., Freeman-Benson B. et Wilson, "Constraint Hierarchies", LISP and Symbolic Computation, an International Journal,5, pp. 223-270, Kluver Academic publisher, 1992.

[Borning98] Borning A., Freeman-Benson B., "Ultraviolet: A Constraint Satisfaction Algorithm for Interactive Graphics", Constraints, Special Issue onConstraints, Graphics, and Visualization, Vol. 3 No. 1, pp. 9-32, April 1998.

[Bulterman00] Bulterman D., GRiNS, on line : http://www.cwi.nl/GRiNS/

[Buchanan93] Buchanan M. C. et Zellweger P. T., "Firefly : Automatic Temporal Layout Mechanisms", Proceedings of the First ACM International Conference on Multimedia, pp. 341-350, ACM Press, Anaheim, Californie, août 1993.

[Byzance00] Byzance, "Manuel de référence de Byzance", Rapport de fin de contrat Géni, mai 2000.

[Carbonneaux99] Carbonneaux Y., Jourdan M., Roisin C., Tardif L. et Villard L., "Architecture détaillée du logiciel Madeus2.1", Rapport de contrat, INRIA Rhône-Alpes, janvier 1999.

[Carcone97] Carcone L., "Formatage spatial dans un environnement d'édition/présentation de documents multimédia", Mémoire, Cnam, décembre 1997.

[Carcone97b] Carcone L., Jourdan M. et Roisin C., "Présentation de documents multimédia basée sur les contraintes", Workshop on Electronic Page Models, LAMPE, 22-23 septembre 1997.

[Carrer97] Carrer M., Ligresti L., Ahanger G. et T.D.C. Little, "An Annotation Engine for Supporting Video Database Population", Multimedia Tools and Applications Vol. 5, No. 3, pp. 233-258, novembre 1997.

[Carrive00] Carrive, J. Roy, P. Pachet, F. Ronfard, R. "A language for audiovisual template specification and recognition", Sixth International Conference on Principles and Practice of Constraint Programming, September 18 - 22, 2000.

[Caseau94] Caseau Y., "Constraint Satisfaction with an Object-Oriented Knowledge Representation Language", Journal of the Applied Intelligence, vol. 4, pp. 157-184, 1994.

[Caseau96] Caseau Y. et Laburthe F., "CLAIRE: Combining Objects and Rules for Problem Solving", Proceedings of the JICSLP'96 workshop on multi-paradigm logic programming, M.T. Chakravarty, Y. Guo, T. Ida eds., TU berlin, 1996

[Chang97] Chang S.-F., Chen W., Meng H.J., Sundaram H. et Zhong D., "VideoQ- An Automatic Content-Based Video Search System Using Visual Cues," ACM Multimedia Conference, Seattle, WA, novembre 1997

[Charman94] Charman P., "A Constraint-based approach for the generation of floor plans". In Proceedings Workshop AAAI'94 on Spatial and Temporal Reasoning, Seattle, 1994.

[Chleq95] Chleq N., "Efficient Algorithms for Networks of Quantitative Temporal Constraints", CONSTRAINTS'95 : the international workshop on constraint-based raisonning, held in conjunction with FLAIRS-95, disponible à : http://www.sci.tamucc.edu/constraints95/home.html,1995.

[Coutaz90] Coutaz J., "Interface homme-ordinateur: conception et réalisation", ED. Dunod informatique , 455 pages, 1990.

[Crilly95] Crilly, "The geometer sketchPad 6.0", Math & Stats, Vol. 6, Num. 2, pp 34.36, mai 1995.

[Dantzig49] Dantzig G. B. "Programming of Interdependent Activities. II. Mathematical Model", Econometrica 17, 200-211, 1949.

[Dantzig54] Dantzig G. B., "The Dual Simplex Algorithm (Notes on Linear Programming: Part VII)" RM-1270, The RAND Corporation, juillet 1954.

[Dechter88] Dechter R. et Dechter A., "Belief Maintenance in Dynamic Constraint Networks", *In: Proceedingsof AAAI-88*, pp. 37-42, St Paul, MN, 1988.

[Dechter91] Dechter R., Meiri I. et Pearl J., "Temporal constraint networks", *Artificial Intelligence*, 49, pp. 61-95, 1991.

[Decouchant96] Decouchant D. et Romero Salcedo M., "Alliance: a Structured Cooperative Editor on the Web", ERCIM News. European Research Consortium for Informatics and Mathematics, num. 25, pp. 18-19, avril 1996.

[DelBimbo96] Del Bimbo A. et Vicario E., "Visual Programming of Virtuals Worlds Animation", IEEE Multimedia, Vol. 3 (Num. 1), pp. 40-49, 1996.

[DHTML98] "Dynamic HTML", disponible à : http://www.dhtml-zone.com/

[Dias00] Dias V., "Intégration de résolveurs de contraintes pour l'édition dans la vue temporelle", Rapport de Maîtrise, Université Joseph Fourrier, septembre 2000.

[DOM98] W3C,"Document Object Model (DOM) Level 1 Specification", W3C Recommendation 1 octobre 1998, disponible à : http://www.w3.org/TR/REC-DOM-Level-1

[DOM2-00a] W3C, "W3C (World Wide Web Consortium) Document Object Model (DOM) Level 2 Specification. W3C Candidate Recommendation, 10 mai 2000, disponible à : http://www.w3.org/TR/DOM-Level-2

[DOM2-00] W3C, "W3C Document Object Model Level 2 Events", T. Pixley, W3C Candidate Recommendation, disponible à :

http://www.w3.org/TR/DOM-Level-2/events.html

[Duluc00] Duluc F., Roisin C., Tardif L. et Villard L., "Un système multimédia complet pour la documentation technique aéronautique", L'Objet, numéro thématique : Objets et multimédia - Éditions Hermès, vol. 6, num. 3, 2000.

[Duluc00b] Duluc F., "Modélisation d'un fond documentaire multimédia : application à la documentation technique aéronautique", Thèse de doctorat, Université Paul Sabatier de Toulouse, octobre 2000.

[Dumas00] Dumas M., Lozano R., Fauvet M.-C., Martin H. et Scholl P.-C., "Orthogonally modeling video structuration and annotation", proceedings of the AAAI Workshop on Spatial and Temporal Granularities, Austin TX (USA), juillet 2000.

[Fargier96] Fargier H., Lang J. et Schiex T., "Mixed constraints satisfaction: a framework for decision problems under uncomplete knowledge", proceedings of the first ACM international conference on multimedia, pp. 273-281, ACM Press, Anaheim, Californie, août 1993

[Furuta88] Furuta R., Quint V. et André J., "Interactively Editing Structured Documents", Electronic Publishing -- Origination, Dissemination and Design, vol. 1, num. 1, pp. 19-44, avril 1988.

[Kim95] Kim M. 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 Francisco, novembre 1995.

[Harada96] Harada K., Hara Y., Tanaka E. et Ogawa R., "Anecdote:A Multimedia Storyboarding System with Seamless Authoring Support", Proceedings of the Fourth ACM Multimedia Conference (MULTIMEDIA'96) (MULTIMEDIA96). Boston, Mass., USA, pp. 341-352, 1996.

[Hardman93] Hardman L., Van Rosusum G. et Bulterman D.C.A. "Structured Multimedia Authoring", Proceedings of the first ACM International Conference on Multimedia, pp. 283-289, Anaheim, août 1993.

[Hauser00] Hauser J. et Rothermel K. "Specification and Implementation of an Extensible Multimedia System" in Proceedings of Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS2000), Springer, 2000.

[HonWai99] Andy Hon Wai C., "Constraint programming in Java with jSolver", Proceedings of PACLP99, The Practical Application of Constraints Technologies and Logic Programming, London, avril 1999.

[Hosobe96] Hosobe H., Matsuoka S., Yonezawa A., "Generalized local propagation: a framework for solving constraint hierarchies", Proceedings of CP 96, Boston, 1996.

[Hosobe98] Hosobe H., Matsuoka S. et Yonezawa A., "HiRise: An Incremental Constraint Solver for Constructing Graphical User Interfaces", Interactive Systems and Software VI---JSSST WISS'98 (M. Yasumura, ed.), vol. 21 dans Lecture Notes in Software Science, Kindai-Kagaku-Sha, pp. 73-82, décembre 1998.

[Hudson95] Hudson et Smith, "Automatic Generation of Starfield Displays Using Constraints",Conference companion on Human factors in computing systems, pp. 202-203, 1995.

[IlogSolver00] IlogSolver, 2000, disponible à : http://www.ilog.com

[MHEG93] ISO/IEC. JTC 1/SC 29, "Codec Representation of Multimedia and Hypermedia Information Objects (MHEG)", Part 1, Comitee Draft 13522-&, juin 1993.

[JAVA00] Sun , "The java langage", disponible à : http://java.sun.com/

[JAVA2D00] Sun , "The java 2D API", disponible à http://java.sun.com/products/java-media/2D/index.html

[JAVA3D00] Sun , "The java 3D API", disponible à : http://java.sun.com/products/java-media/3D/index.html

[JMF00] Sun, "The Java Media Framework 2.0", disponible à : http://java.sun.com/products/java-media/jmf/index.html

[Jourdan97] Jourdan M., Layaïda N. et Sabry-Ismail L., "Time Representation and Management in MADEUS: an authoring environment for multimedia documents", Multimedia Computing and Networking 1997, M. Freeman, P. Jardetzki, H.M. Vin, SPIE 3020, San-Jose, février 1997.

[Jourdan97a] Jourdan M., Roisin C. et Tardif L., "Visualization of constrainted-based Temporal Scenarios in a Multimedia Authoring tool", Rapport interne num. 3284, INRIA, octobre 1997.

[Jourdan97b] Jourdan M., Roisin C. et Tardif L., "User Interface of a new generation of authoring environment of multimedias documents", Proceeding of the Third ERCIM Workshop on User Interfaces for All, Strasbourg (France), 3-4 novembre 1997.

[Jourdan 98a] Jourdan M., Layaïda N., Roisin C., Sabry-Ismail L. et Tardif L., "Madeus, an Authoring Environment for Interactive Multimedia Documents", ACM Multimedia'98, pp. 267-272, ACM, Bristol (UK), septembre 1998.

[Jourdan 98b] Jourdan M., Roisin C. et Tardif L., "Édition et Visualisation Interactive de Documents Multimedia", Electronic Publishing'98, St Malo, France, avril 98.

[Jourdan98c] Jourdan M., Roisin C. et Tardif L., "Constraints Techniques for Authoring Multimedia Documents", ECAI 98 Workshop on Constraints for artistic applications, Brighton (UK), août 1998.

[Jourdan98d] Jourdan M., Roisin C. et Tardif L., "Multiviews Interfaces for Multimedia Authoring Environments", Proceedings of the 5th Conference on Multimedia Modeling, pp. 72-79, IEEE Computer Society, Lausanne, octobre 1998.

[Jourdan99] Jourdan M., Roisin C., Tardif L. et Villard L., "Authoring SMIL documents by direct manipulations during presentation", World Wide Web, Balzer Science Publishers, vol. 2, num. 4,

décembre 1999.

[Jourdan00] Jourdan M., Roisin C. et Tardif L., "A Scalable Toolkit for Designing Multimedia Authoring Environments", numéro spécial : Multimedia Authoring and Presentation: Strategies, Tools, and Experiences, de Multimedia Tools and Applications Journal, Kluwer Academic Publishers, à paraître 2000.

[Jourdan00b] Jourdan M., Roisin C. et Tardif L., "Constraints Techniques for Authoring Multimedia Documents", Constraints Journal, Kluwer Academic Publishers, à paraître 2000.

[Helm95] Helm R., Huynh T., Marriott K. et Vlissides J., "An object-oriented architecture for constraints based graphical editing", C. Laffra , E. Blake, V. de Mey et X. Pintado, éditeurs, Object oriented Programming for Graphics, pp. 217-238, Springer-verlag, 1995.

[Hosobe96] Hosobe H., Matsuoka S., Yonezawa A., "Generalized local propagation: a framework for solving constraint hierarchies", Proceedings of CP 96, Boston, 1996.

[Laborde95] Laborde J.M. et Laborde C., "Des connaissances abstraites aux réalités artificielles, le concept de micromonde cabri", Environnements interactifs d'apprentissage avec ordinateur (tome 2), Eyrolles Paris, pp. 29-41, 1995.

[Laforge00] Laforge M., "Gestion prédictive et réactive de la qualité de service pour les documents multimédia temporisés", Mémoire, Cnam, décembre 2000.

[Layaïda96] Layaïda N. et Sabry-Ismaïl L., "Maintening Temporal Consistency of Multimedia Document Using Constraints Networks", Proceedings of the International Conference on Multimedia Computing and Networking, San José, CA, USA, janvier 1996.

[Layaïda97] Layaïda N., "Madeus: système d'édition et de présentation de documents structurés multimédia", Doctorat Informatique, Université Joseph Fourrier, 1997.

[Layaida99] Layaïda N., "Adaptabilité : pistes d'étude pour la définition d'une infrastructure d'accès au contenu multimédia pour des machines hétérogènes", Rapport de contrat, INRIA Rhône-Alpes, octobre 1999.

[Mackinlay91] Mackinlay J.D., Robertson G.G. et Card S.D., "The Perspective Wall : Detail and context smoothly integrated", Proceedings of Chi'91 Human Factors in Computing Systems, pp. 173-177, Addison Wesley, 1991.

[Mackworth77] Mackworth A.K., "Consistency in Networks of Relations", *Artificial Intelligence*, 8 (1), pp. 99-118, 1977.

[Mackworth85] Mackworth A.K. et Freuder E. C., "The complexity of some Polynomial Consistency Algorithms for Constraints Satisfaction Problems", Artificial Intelligence, 25(1), pp. 65-74, 1985.

[Macromédia-Director00] Macromedia, "Director : a script langage", disponible à : http://www.macromedia.com/

[Macromédia-Flash00] Macromedia, "Flash", disponible à : http://www.macromedia.com/software/flash/

[Macromedia-Lingo00] Macromedia, "Lingo : a script langage", disponible à : http://www.macromedia.com/

[Maloney89] Maloney J.H., Borning A. et Freeman-Benson B.N., "Constraint technology for user-interface construction in ThingLab II", SIGPLAN Notices vol. 24, issue 10, pp. 381-388, 1989.

[Maple98] Maple, Reference Manual, 1998.

[Marriott98] Marriott K. et Struckey P., "Introduction to Constraints Logic Programming", Mit Press, 1998.

[Marriott98b] Marriott K., Sen Chok S. et Finlay A., "A Tableau Based Constraint Solving Toolkit for Interactive GraphicalApplication ", 4th International Conference on Principles and Practice of Constraint, Pise 1998.

[Martin99] Martin H. et Mulhem P., "A Comparison of XML and SMIL for on the fly generation of Multimedia Documents from Databases", 6th International Conference on Information Systems Analysis and Synthesis (ISAS2000), Orlando, USA, juillet 2000.

[Meiri92] Meiri I, "Temporal Reasoning : A constraint-based Approach", Cognitive Systems Laboratory, University of California, Los Angeles, CA90024, janvier 1992.

[Meurisse99] Meurisse M., "Edition de document multimédia, vue temporelle", Mémoire de fin d'étude, Université de Namur, 1999.

[Microsoft-PowerPoint00d] Microsoft, "PowerPoint", disponible à : http://www.microsoft.com/office/powerpoint/

[MHML98] Alcatel, "MHML : Langage de description de document multimédia", rapport interne, 1998.

[Mohr86] Mohr R. et Henderson T., "Arc and path consistency revisited", *Artificial Intelligence*, 28, pp. 225-233, 1986.

[Mohr99] Mohr R., Roisin C., Tran Thuong T. et Villard L., Rapport de description XML des structures de vidéo, Rapport de contrat, INRIA Rhône-Alpes, mars 1999.

[Morse99] L. Morse, "Evaluation of Visual Information Browsing Displays", Doctor of Philosophy,Graduate Faculty of the Department of Information Science and Telecommunications of the School of Information Sciences, University of Pittsburgh, 1999.

[Myers97] Myers B.A., Borison E., Ferrency A., McDaniel R., Miller R.C., Faulring A., Kyle B., Doane P., Mickish A., Klimovitski A., "The Amulet V3.0 Reference Manual", March, 1997 CMU-CS- 95-166- R2 CMU- HCII- 95- 102- R2, mars 1997.

[Navarro00] Navarro P., "Edition de document multimédia : SMIL", Mémoire, Cnam, décembre 2000.

[Pernin96] Pernin, J. P. et Meunier C., "MELISA Core, environnement de production de simulations", Grenoble (France), CLIPS-IMAG, 1996.

[Quint87] Quint V., "Une approche de l'édition structurée des documents", Thèse d'État, Université Scientifique, Technologique et Médicale de Grenoble, Mai 1987.

[Quint99] Quint V. et Roisin C., "Thot", disponible à : http://www.inrialpes.fr/opera/Thot.fr.html

[Real98] Real Netwoks , Real Netwoks G2, on line :http://www.real.com/g2/index.html

[Real99] Real Netwoks , Smil Wizard in G2 Authoring Kit, disponible à :http://www.real.com/products/tools/authkit/index.html.

[Robertson91] Robertson G.G., Mackinlay J.D. et Card S.D., "Cone trees: Animated 3D visualizations of hierarchical information"", Proceedings of the CAM SIGSHI conference on Human Factors in Computing Systems, pp. 189-194, ACM PRESS, New York, 1991.

[Roisin00] Roisin C., Tran_Thuong T. et Villard L., "A Proposal for a Video Modeling for Composing Multimedia Document", International Conference on Multimedia Modeling 2000 (MMM2000), Nagano, novembre 2000.

[Saade97] Saade D. C. M., Soares L. F. G., Costa F. R. et Souza Filho G. L., "Graphical Structured-Editing of Multimedia Documents with Temporal and Spatial Constraint", Proceedingsof the 4th Conference on Multimedia Modelling, pp. 279-295, World Scientific Publishing, Singapore, 18-19 novembre 1997.

[Sabry98] Sabry-Ismaïl L. et Guetari R., "Le modèle Objet Madeus", L'Objet : Les Représentations par Objets en Conception, Editions Hermès, vol. 4, num. 2, 1998.

[Sabry99] Sabry-Ismaïl L., "Schéma d'exécution pour les documents multimédia distribués", Doctorat d'informatique, Université Joseph Fourrier, janvier 1999.

[Sausage98] Sausage, Smil Composer, on line :http://www.sausage.com/supertoolz/toolz/stsmil.html.

[Sannella92] Sannella M. et Borning A., "Multi-Garnet: Integrating Multi-Way Constraints with Garnet", UW tech report 92-07-01, 1992.

[Sannella93] Sannella M., Maloney J., Freeman-Benson J. et Borning A., "Multi-way versus One-way constraints in user interfaces: Experience with the DelatBlue algorithm", Software-Pratice and Experience, Vol. 32(5), pp. 529-566, mai 1993.

[Sannella95] Michael Sannella, "The Skyblue Constraint Solver and Its Applications", Vijay Saraswat and Pascal van Hentenryck, editors, Proceedings of the 1993 Workshop on Principles and Practice of Constraint Programming, pp. 385-406, MIT Press, 1995.

[Sarkar94] Sarkar M. et Brown M., "Graphical Fisheye Views", Communication of the ACM, Vol. 37 (Num. 12), pp. 73-84, 1994.

[Séraphine01] Séraphine F., "Édition collaborative sur le Web et support du travail collaboratif", Doctorat d'informatique, Université Joseph Fourrier, vers 2001.

[SMIL98] W3C, "Synchronized Multimedia Integration Language (SMIL) 1.0" P. Hoschka. W3C Recommendation 15 juin 1998, disponible à : http://www.w3.org/TR/REC-smil.

[SMIL-DOM00] W3C, "SMIL DOM : Synchronized Multimedia Integration Language Document Object Model (DOM)". W3C Working Draft, work in progress. disponible à : http://www.w3.org/TR/smil-boston-dom/

[SMIL2-00] W3C, "SMIL-2.0 : Synchronized Multimedia Integration Language (SMIL) Boston Specification". W3C Working Draft, work in progress. disponible à : http://www.w3.org/TR/smil-boston/

[Song99] Song J., Ramalingam G., Miller R. et Byoung-Kee Yi, "Interactive authoring of multimedia documents in a constraint-based authoring system", Multimedia Systems 7 , 424-437, 1999.

[SVG00] W3C, "SVG : Scalable Vector Graphics (SVG) 1.0 Specification". W3C Working Draft, work in progress. disponible à :

http://www.w3.org/TR/SVG

[Tardif97] Tardif L., "Visualisation du scénario temporel d'un document multimédia", Rapport de DEA, INPG-Universite Joseph Fourrier, juin 1997 .

[Tardif97b] Tardif L., "Traitement et visualisation de la flexibilité du scénario temporel d'un document multimédia", Rapport de Magistère, Université Joseph Fourrier, septembre1997.

[Tardif00] Tardif L., Bes F. et Roisin C. ,"Constraints for multimedia documents", conférence PACLP2000, Manchester, avril 2000

[Takahashi95] Takahashi S., Matsuoka S., Miyashita K., Hosobe H., Yonezawa A. et Kamada T., "TRIP : A Constraint-based Approach for Visualization and Animation", international Workshop on Constraints for Graphics and Visualization, Cassis (France), 1995.

[Tissot00] Tissot M., "Environnement d'édition de Workflow", Rapport de Maîtrise, Université Joseph Fourrier, septembre 2000.

[Tufte83] Tufte R, "The Visual Display of Quantitative Information", Graphics Press, Cheshire, CT, 1983.

[Uginet00] Uginet A et Roisin C., "Etude de faisabilité pour un environnement auteur MHML avec la boîte à outils Kaomi", Rapport de contrat Alcatel, 1999.

[Vazirgianis98] Vazirgianis M., "Interactive Multimedia Documents", Lectures Notes in Computer Science, Ed. Goos G., Hartmanis J. et van Leeuwen J., 1998.

[Verfaillie95] Verfaillie G. et Schiex T., "Maintien de solution dans les problèmes dynamiques de satisfaction de contraintes: bilan de quelques approches", *Revue d intelligence artificielle*, volume 9, n°3, pp. 269-309, 1995.

[Vidal95] Vidal T., "Le temps en planification et ordonnancement: vers une gestion complète et efficace de contraintes hétérogènes et entachées d'incertitude", Thèse de doctorat, Université Paul Sabatier de Toulouse, septembre 95.

[Vidal97] Vidal T., "A preliminary characterization of Dynamics Controllability in contingent Temporal CSPs", Rapport Interne,1997

[Vilard00] Villard L., Roisin C. et Layaïda N., "A XML-based multimedia document processing model for content adaptation", Digital Documents and Electronic Publishing (DDEP00), LNCS, septembre 2000.

[VRML99] "Official and complete specification of the Virtual Reality Modeling Language, (VRML)", ISO/IEC DIS 14772, disponible à :

http://www.web3d.org/fs_specifications.htm .

[Wilcox97] Wilcox E. M., Atwood J.W., Burnett M. M., Cadiz J. J. et Cook C. R., "Does continuous visual feedback aid debugging in direct-manipulation programming systems?", conference proceedings on Human factors in computing systems, pp. 258-265, 1997.

[Wirag95] Wirag S., Wahl T. et Rothermel K., "Tiempo : An Authoring and Presentation System for

Interactive Multimedia", Fakultätsbericht 5/95, Technical Report, Fakultät Informatik, University of Stuttgart, Germany, 1995, disponible à : http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/tiempo.engl.html

[Xerox00] Xerox, Hyperbolic Tree, disponible à :http://www.inxight.com/demos/ht_walk_thru/index.html

[XML99] W3C, "Extensible Markup Language (XML)" disponible à : http://www.W3C.org/XML/