



## 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).

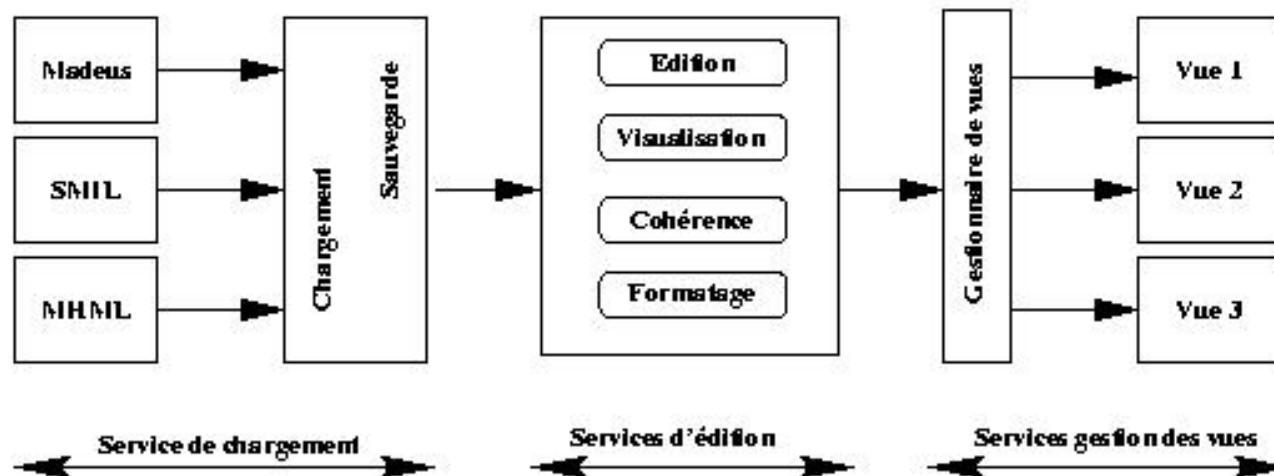


Figure IV-2 : Structures et services fournis par Kaomi

### 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.

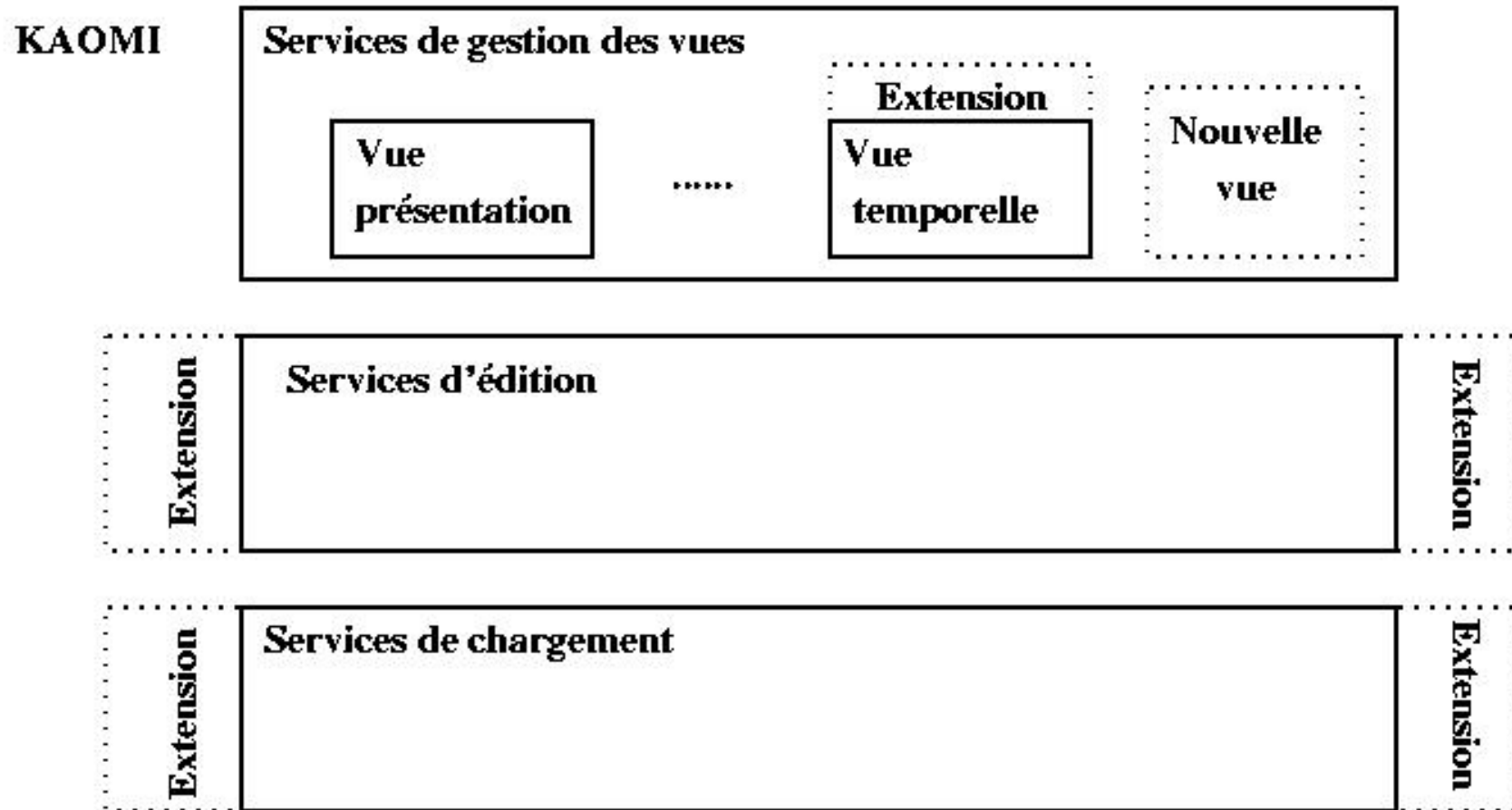
L'ensemble de ces vues, ainsi que leurs mécanismes de construction et de synchronisation, seront présentés dans la section 9. La vue vidéo structurée qui est l'objet d'une thèse en cours sera, elle présentée dans le chapitre VI.

## 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 fournit 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.



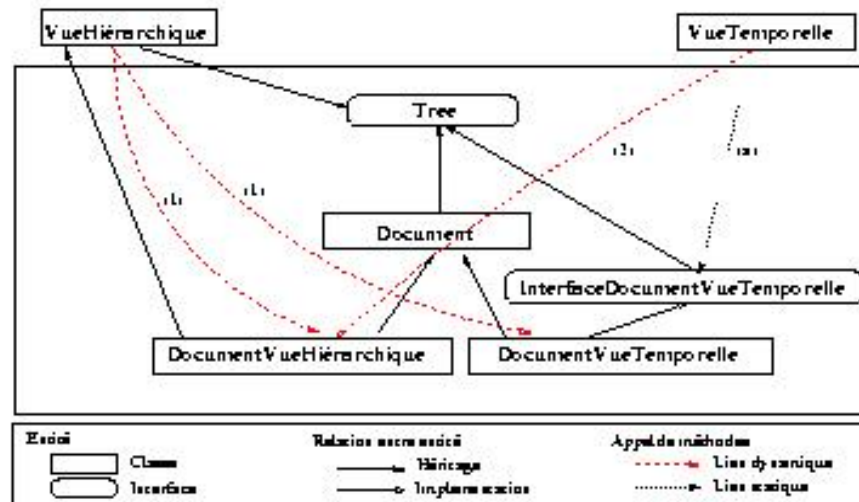


Figure IV-5 : Exemple d'utilisation du mécanisme d'interface de Kaomi

### 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.

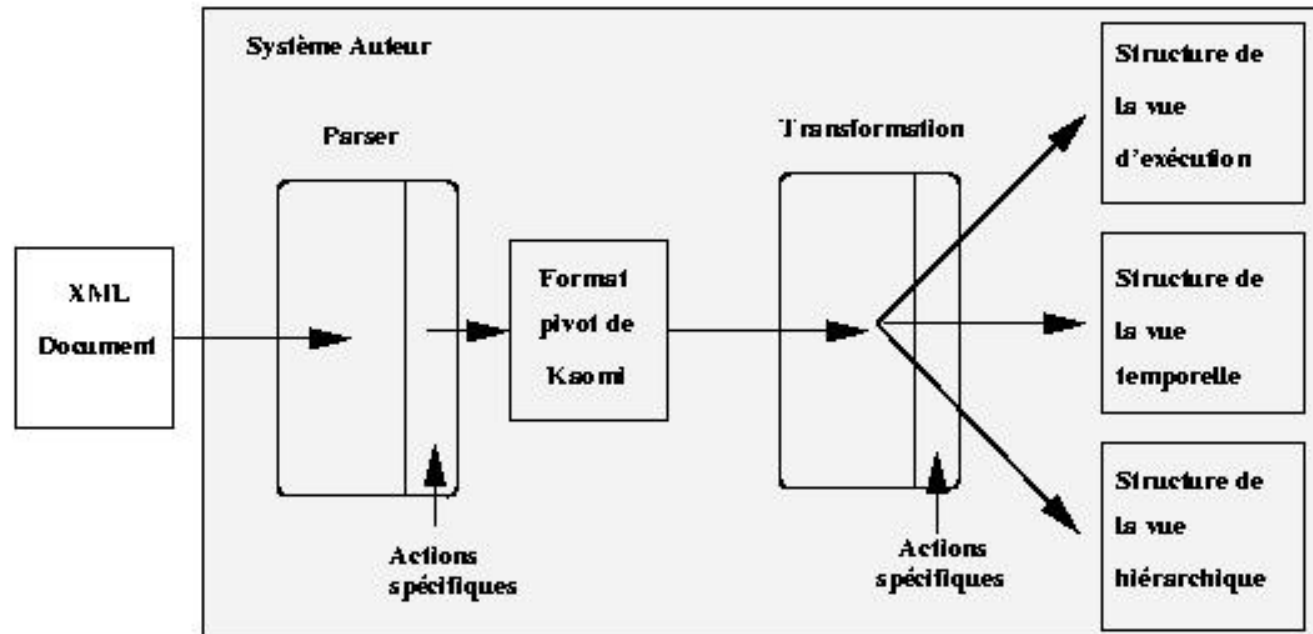


Figure IV-9 : Mécanisme de création des vues

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 notamment 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.

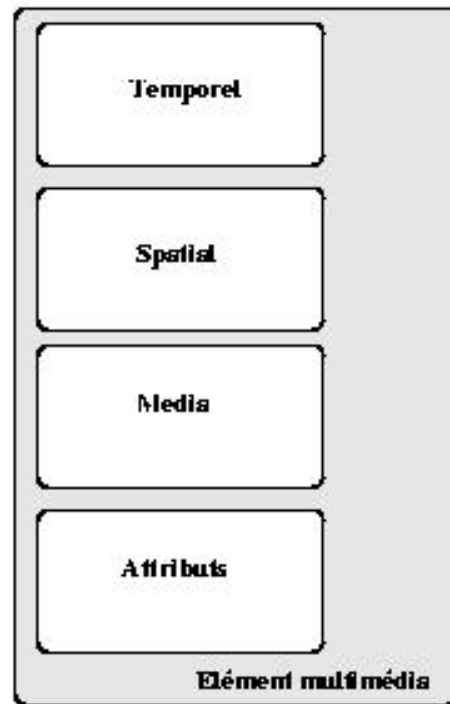
Dans cette perspective, la structure de données pour représenter les documents multimédias est basée essentiellement sur un élément multimédia qui contient les informations temporelles, spatiales, de navigation (hyperliens), le type ainsi que les informations concernant les médias (objets son, image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

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.

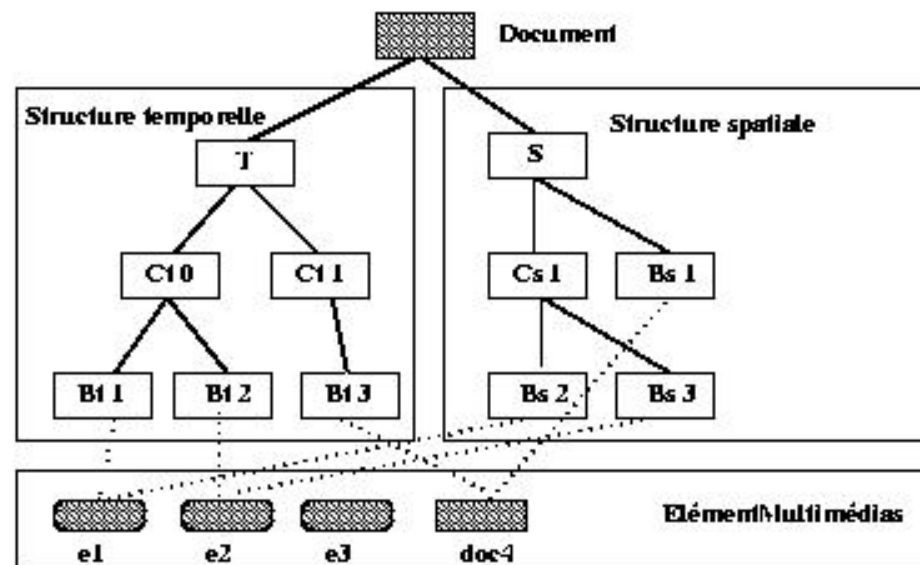


Figure IV-11 : Structure d'un document dans Kaomi

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



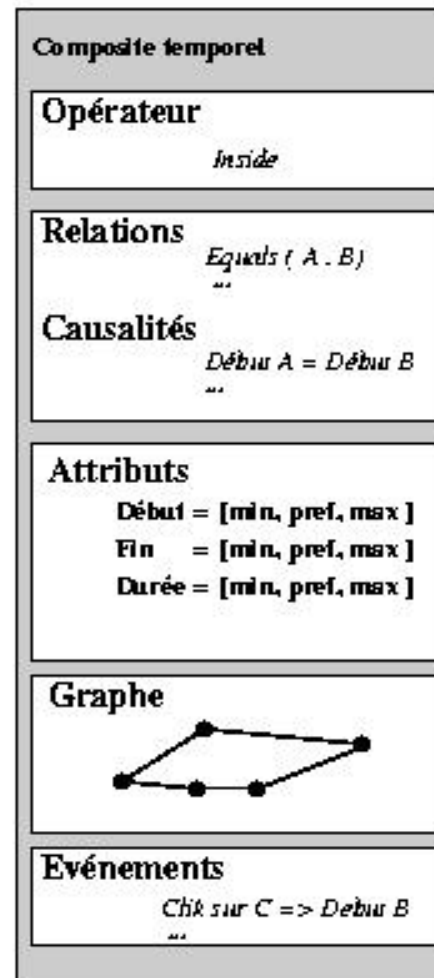
regrouper un ensemble d'objets, les objets basiques sont les feuilles de la structure et sont liés aux objets multimédias. Nous allons présenter de manière plus précise la structure temporelle dans la section 7.

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'instantanées 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 ( $\Delta$ ).
- 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)	$\Delta$	-12s
Début (Objet Temporel B)	$\Delta$	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.nbrRélem = 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 événements 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 =  $\{t_1, t_2, \dots, t_n\}$  avec  $n$  infini, et  $t_i$  une variable.

Conditions =  $C_1 \& C_2 \& C_3$

Actions =  $\{A_1, A_2, A_3\}$

**Traduction en réléms :**

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 = Média1
Destination = Média2
NbOccurrences = 1
Dates d'occurrences = {14}
Conditions = Fin(Média2)
Actions = Démarrer(Média1) }

```

sera traduit par le rélem suivant :  $Fin(A) \Rightarrow Début(B)$ . 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 *EI* 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 comportements 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 insère 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 *equals* entre les deux objets, relation qui sera traduite en rélems par le même mécanisme.



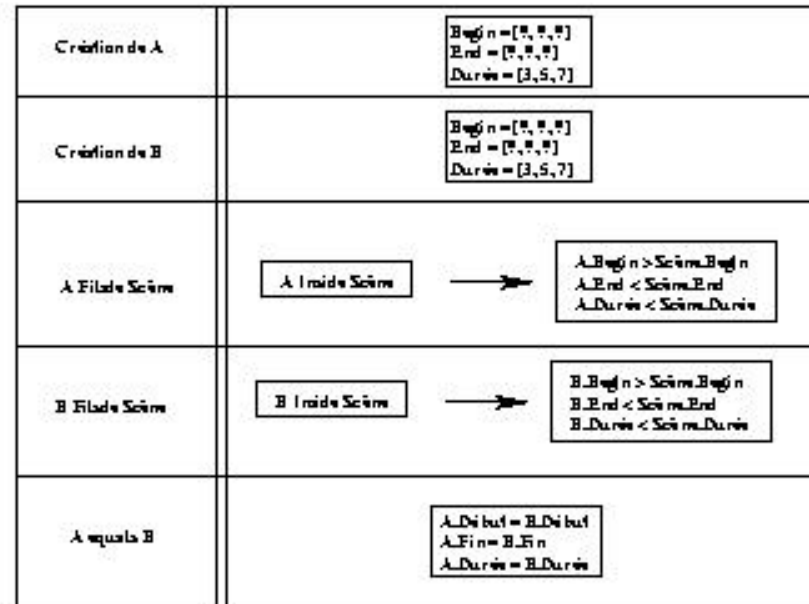


Figure IV-17 : Mécanisme de mise à jour des rélems

## 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.


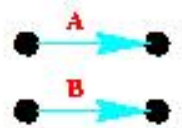
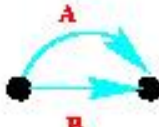
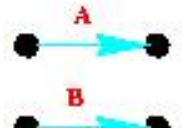

Etat initial	Etat final	Actions
		<b>Création d'un objet</b>
		<b>Insertion de la relation</b> <b>A equals B</b> $Debut(A) = Debut(B)$ $Fin(A) = Fin(B)$
		<b>Insertion d'une relation</b> <b>A before B</b> $Fin(A) < Debut(B)$

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.

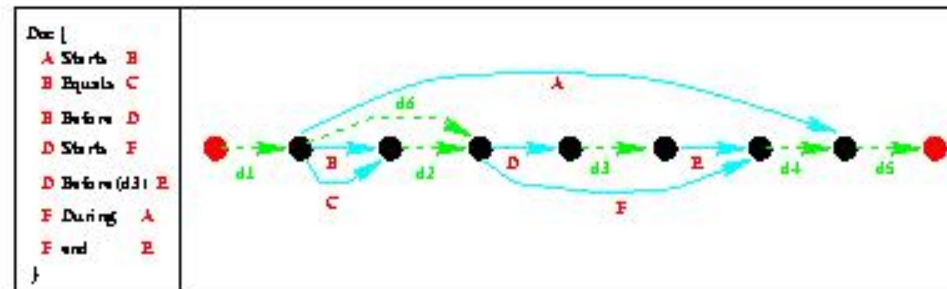


Figure IV-20 : Graphe temporel résultant d'une spécification

### 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  $Fin(A) \Rightarrow Fin(B)$  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 : <ul style="list-style-type: none"> <li>● un arc qui représente la durée prévue de l'objet auquel appartient <math>t_2</math></li> <li>● un délai qui relie la fin (ou le début) prévue (sans interruption) avec la fin (ou le début) réelle de l'objet</li> </ul> L'arc initial représente maintenant la durée réelle de l'objet.
$t_1 \Leftarrow t_2$	Insertion de deux arcs fictifs : <ul style="list-style-type: none"> <li>● un arc qui représente la durée prévue de l'objet auquel appartient <math>t_1</math></li> <li>● un délai qui relie la fin (ou le début) prévue (sans interruption) avec la fin (ou le début) réelle de l'objet</li> </ul> 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.


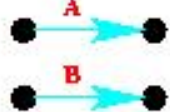
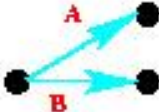

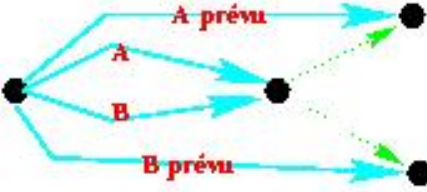
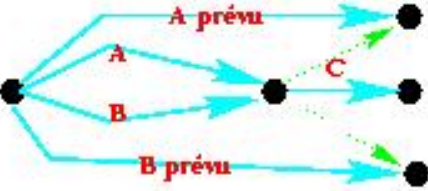
Etat initial	Etat final	Actions
		Création d'un objet
		Insertion de la relation A parmi B $Début(A) = Début(B)$
		$Fin(A) \Rightarrow Fin(B)$
		$Fin(B) \Rightarrow Fin(A)$
		Insertion de la relation A meet C $Fin(A) = Début(B)$

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 tempore. 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.

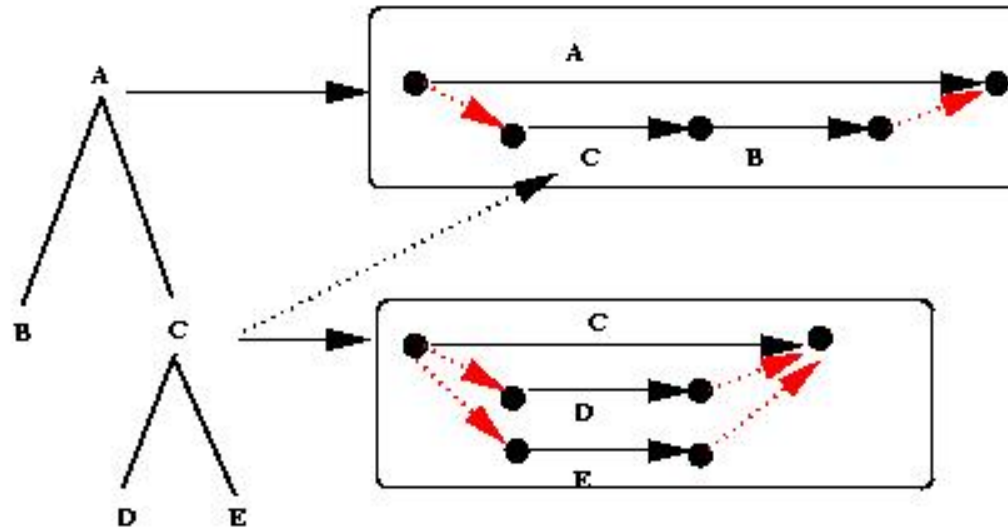


Figure IV-23 : Construction de graphes hiérarchiques

## 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.

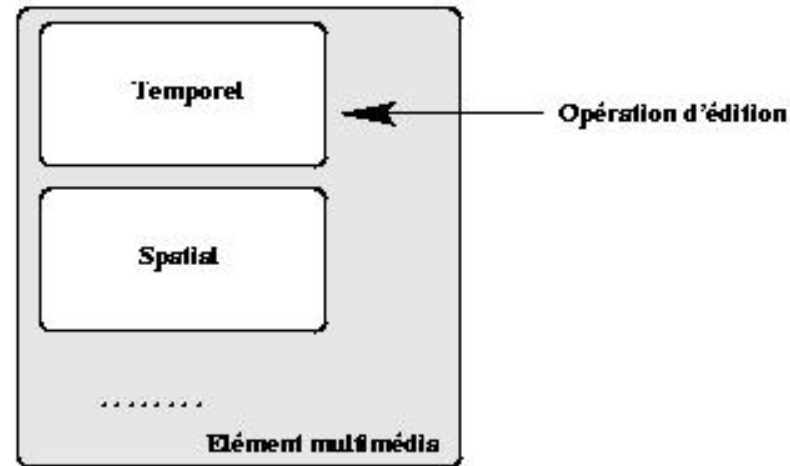
## 8.2 L'édition d'attributs dans Kaomi

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 insère 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.

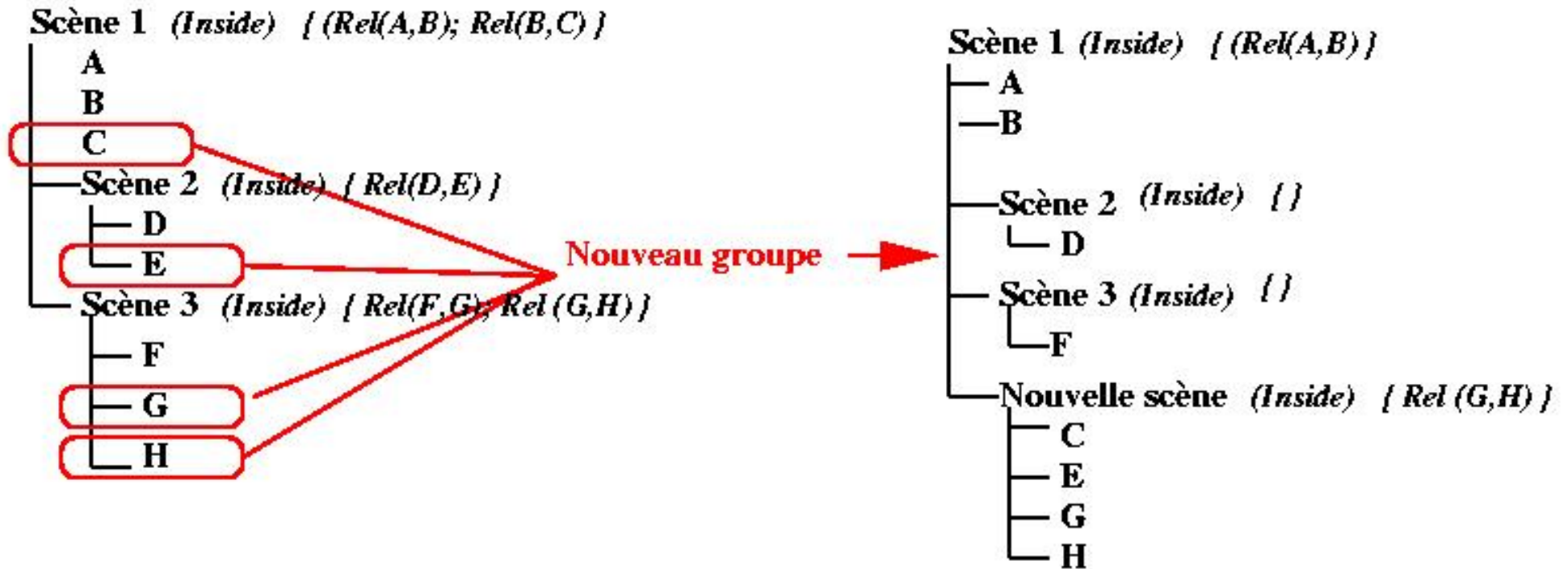


Figure IV-25 : Exemple de création de groupe

## 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.

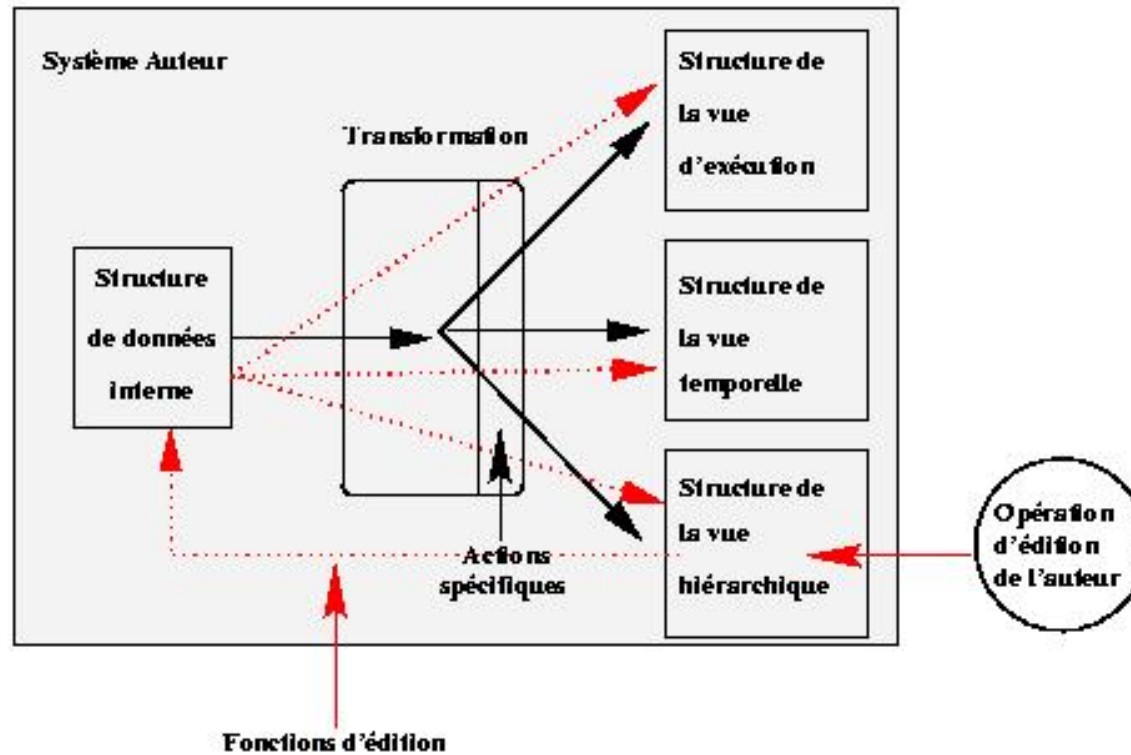


Figure IV-26 : Mécanisme d'édition dans les vues

## 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. Toute 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.

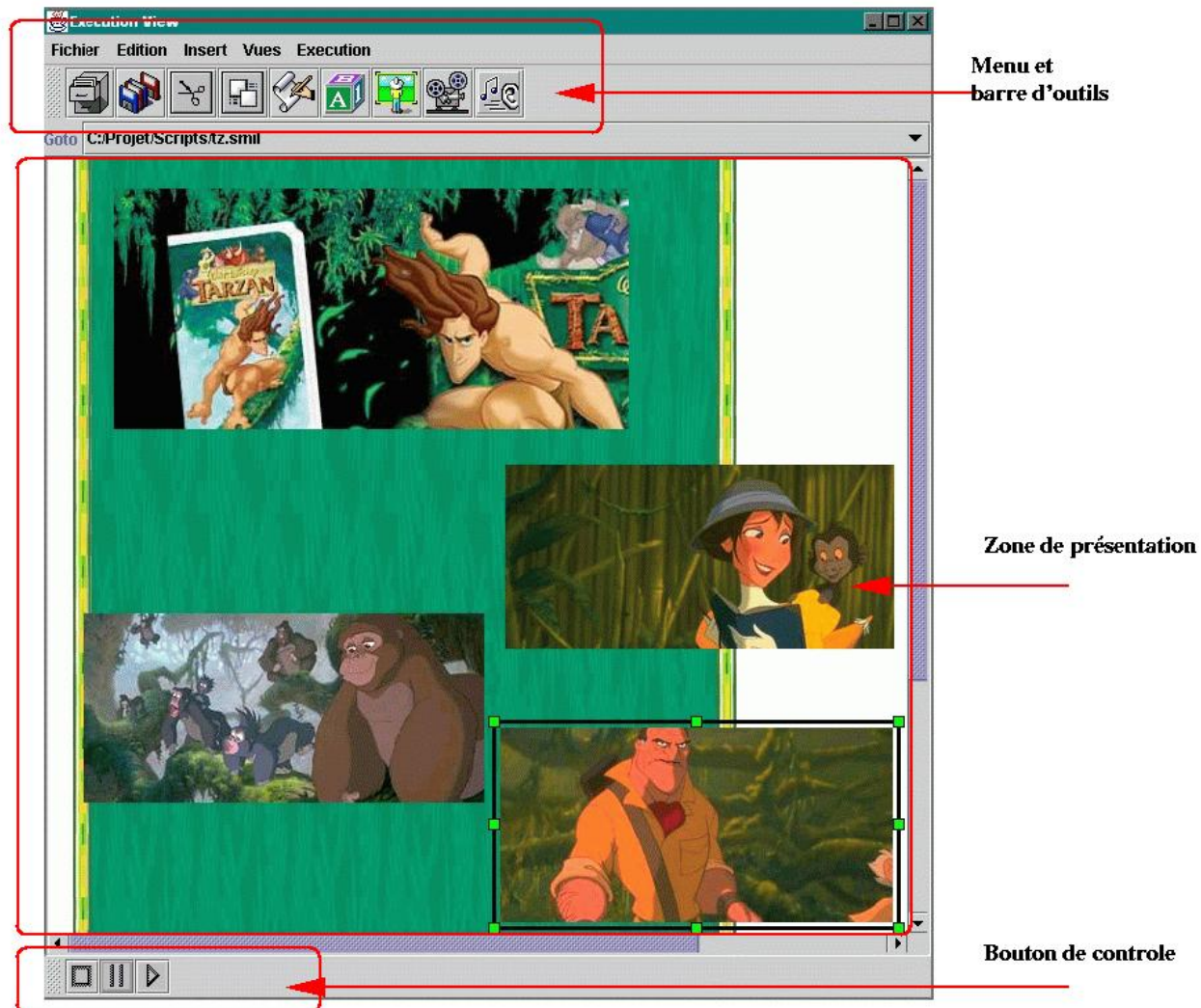
## 9.3 La vue de présentation

La vue de présentation (Figure IV-27) est celle qui implémente les services de présentation du document, et qui fournit des services tels que jouer / stop / pause / reprend. Cette vue permet aussi d'éditionner 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 gèrera 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,...

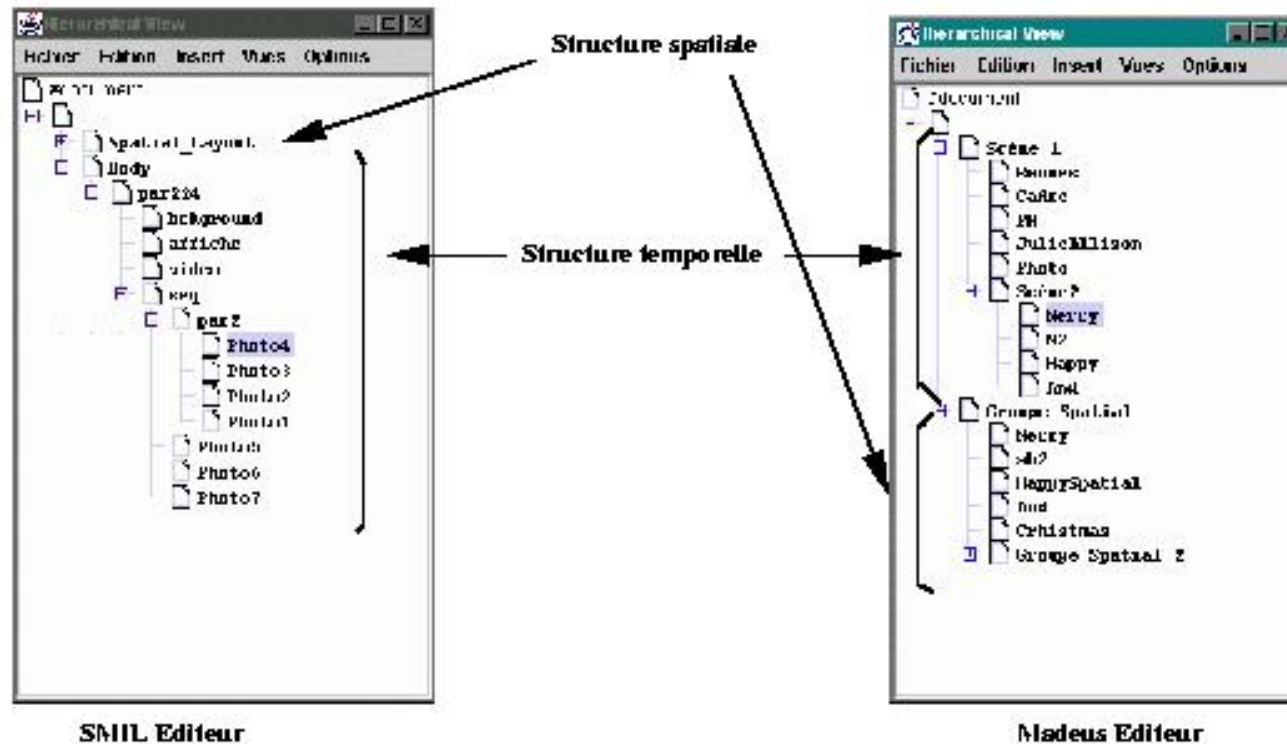


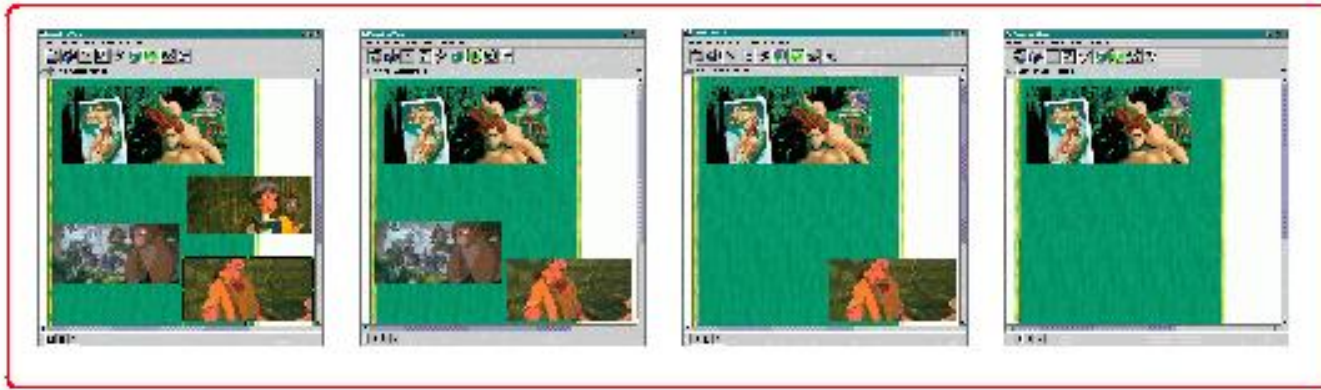
Figure IV-28 : Vue hiérarchique

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.

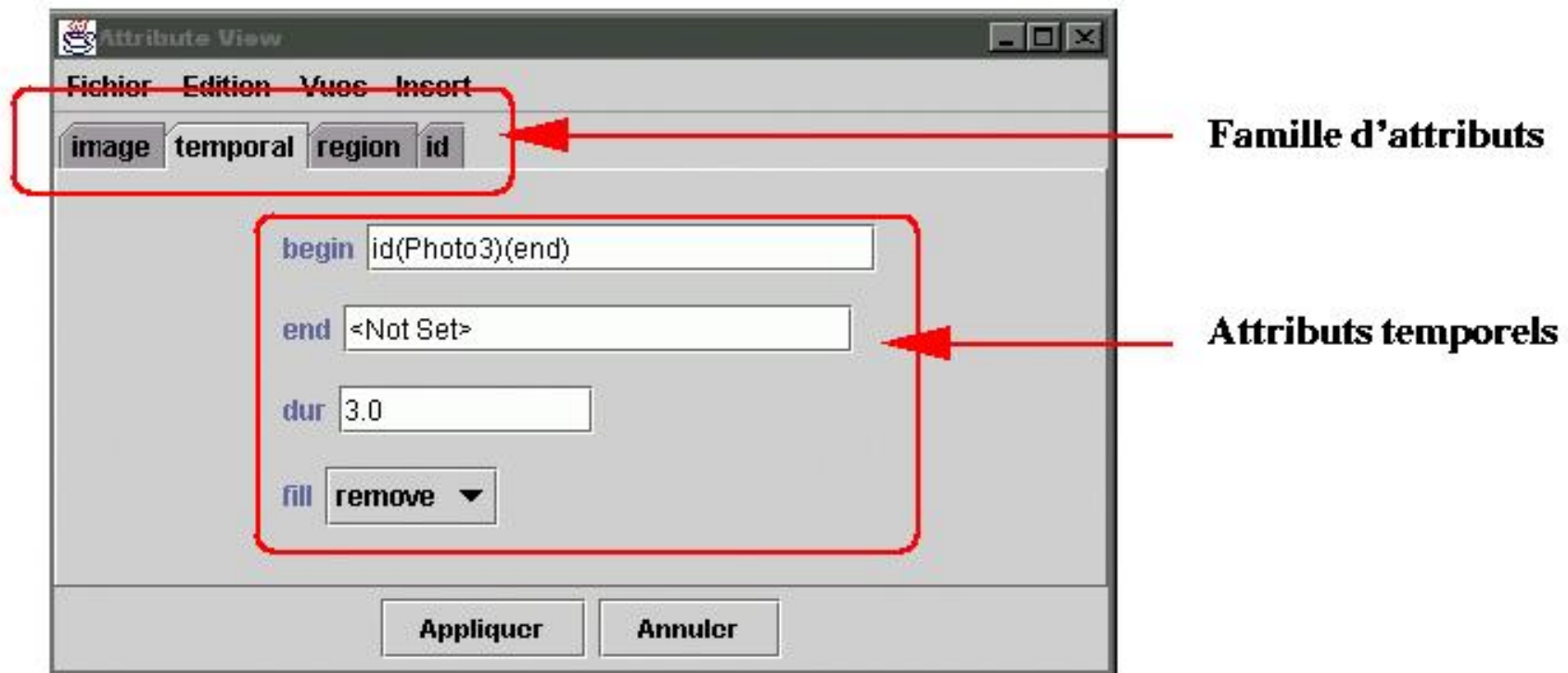


Figure IV-30 : Visualisation des attributs d'un objet

## 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 [Tuft83], 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.

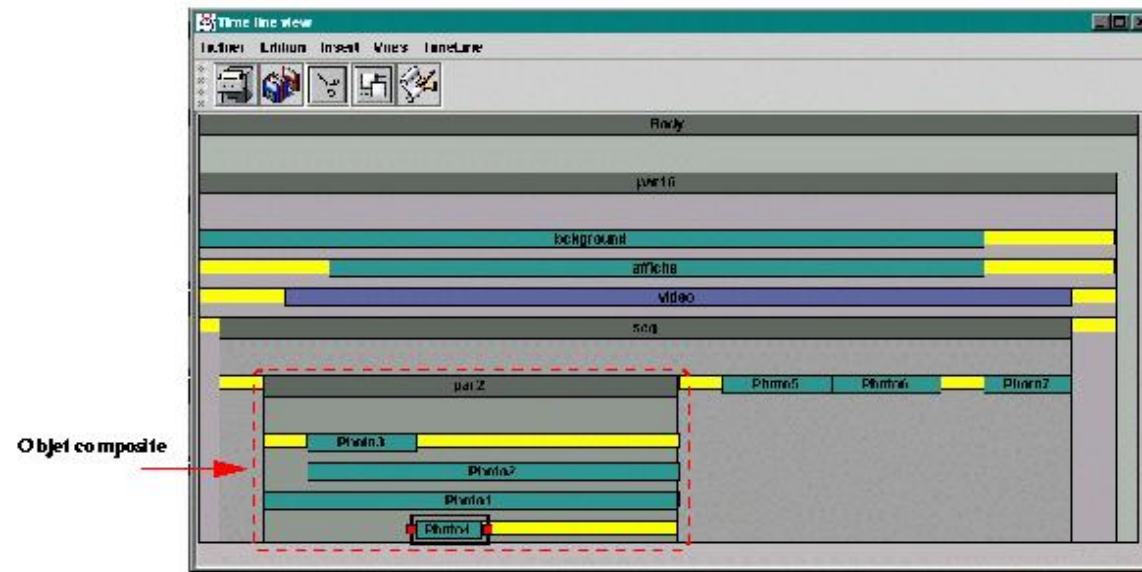
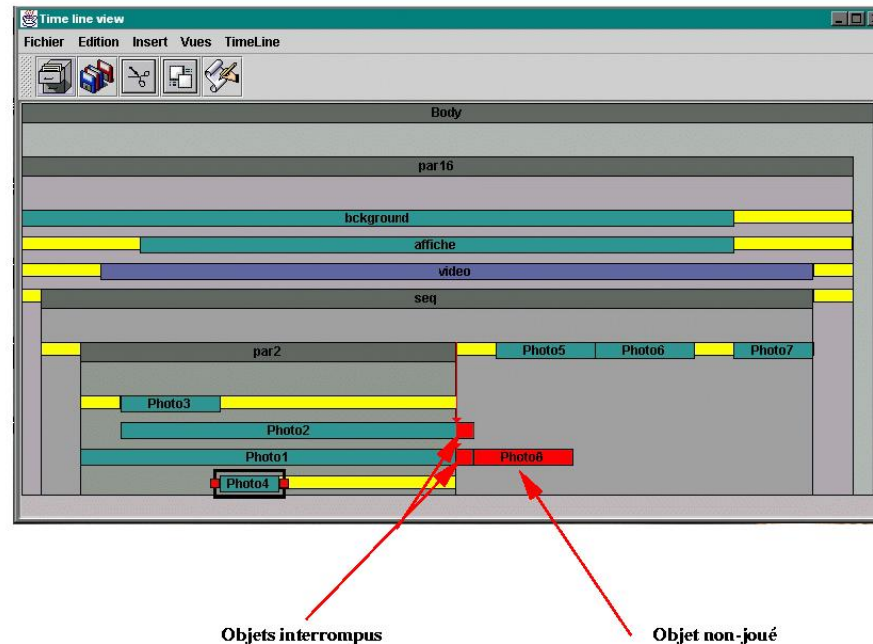


Figure IV-31 : Vue temporelle

### Opérationnel :

- *Les informations d'édition* : le mécanisme d'édition directe dans la vue temporelle qui est réalisé en grande partie à l'aide de solveurs 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.