HAL
open science

Modèle d'édition de document multimédia Mémoire de fin d'études Master d'Informatique

Duc Bao Le

To cite this version:

Duc Bao Le. Modèle d'édition de document multimédia Mémoire de fin d'études Master d'Informatique. 2008. inria-00232724

HAL Id: inria-00232724

https://inria.hal.science/inria-00232724

Preprint submitted on 1 Feb 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT DE LA FRANCOPHONIE POUR L'INFORMATIQUE

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA RHÔNE-ALPES

Modèle d'édition de document multimédia

Mémoire de fin d'études Master d'Informatique

Étudiant : **Duc Bao LE**

Sous la direction de :
Cécile ROISIN, *Professeur*
et
Jan MIKÁČ, *Ph.D*

Grenoble, Septembre 2007

Remerciement

Pour commencer, je tiens à remercier Cécile ROISIN pour m'avoir accueilli dans le projet LimSee3, de l'équipe WAM, à l'INRIA Rhône-Alpes.

Je remercie tout particulièrement Cécile ROISIN et Jan MIKÁČ pour m'avoir guidé et soutenu durant ce stage et pour leurs nombreuses relectures du mémoire.

Résumé

Les travaux de ce stage consistent à étudier les techniques d'édition de document multimédia, le modèle d'édition de LimSee3, les propositions et les expérimentations d'exportation de document LimSee3 vers les formats de présentation multimédia. Nous présentons les différents modèles d'édition qui sont employés par les outils existants. En plus, nous examinons d'exportation de document LimSee3 vers le format de document multimédia comme SMIL et le format qui ne supporte pas directement d'expression temporelle comme XHTML. En fait, le processus d'exportation est complexe en raison de la diversité de format de présentation de document multimédia. Nous proposons deux approches d'exportation ; l'un utilise Java et l'autre utilise un format intermédiaire avec les feuilles XSL. Le résultat est un cadre d'application offert au service d'exportation et les exportateurs de document LimSee3 vers de formats SMIL, XHTML+CSS+JavaScript, et un ordonnanceur en JavaScript qui permet de rajouter les scénarios temporels aux documents XHTML.

Table de matières

1. Introduction 5
1.1 Contexte 5
1.2 Cadre de travail 6
1.3 Plan du mémoire 6
Première partie : Analyse du sujet 7
2. Analyse du sujet 8
2.1 Type d'application 9
2.2 Format de document multimédia 9
2.3 Template 10
2.4 Synthèse 11
Deuxième partie : État de l'art 12
3. Technologie XML 13
3.1 Navigation de document XML 13
3.2 Langages de Schéma 14
3.3 Transformation de documents XML 14
3.4 Synthèse 15
4. Modèle de document multimédia 15
4.1 Synchronized Multimedia Integration Language 16
4.2 Flash 17
4.3 MPEG-4 17
4.4 Scalable Vector Graphics 18
4.5 Synthèse 19
5. Modèle d'édition de documents multimédia 20
5.1 Fonctions d'édition 21
5.2 Typologie des systèmes d'édition 22
5.3 Synthèse 25
6. Modèle de Document LimSee3 26
6.1 Structure 26
6.2 Template 27
6.3 Caractéristiques 29
6.4 Exemple 29
6.5 Conclusion 31
7. Architecture de LimSee3 31
Troisième partie : Contribution 33
8. Exportation 34
8.1 Introduction 34
8.2 Étude de l'exportation de LimSee3 35
8.3 Approches possibles pour l'exportation 42
9. Implémentation 46
9.1 Architecture 46
9.2 Exportation vers SMIL 47
9.3 Exportation vers XHTML+JavaScript+CSS 48
9.4 Exportation avec un template 53
10. Résultats 54
11. Conclusion 54

12. Perspective 55
Bibliographie 57
Article 57
Site Web 58
Annexe 60
Annexe 1: Le DTD d'ordonnancement 60
Annexe 2: Le DTD de format intermédiaire 62

Table de figures

Figure 1: Le processus de création de document multimédia 8
Figure 2: Une structure d'arbre temporel de document (Bulterman 2005) 22
Figure 3: Le timeline d'un document (Bulterman 2005) 23
Figure 4: Un document basé sur le graphe (Bulterman 2005) 24
Figure 5: Le schéma du modèle de document LimSee3 27
Figure 6: La représentation de document par l'axe temporel 31
Figure 7: L'architecture de LimSee3 32
Figure 8: Le processus d'exportation 34
Figure 9 : La structure de l'exemple (Liste 1) 36
Figure 10: La correspondance entre le document LimSee3 et le document SMIL 37
Figure 11: La structure de slideshow 40
Figure 12: L'exportation avec la structure interne de LimSee3 43
Figure 13: L'exportation par un format intermédiaire 44
Figure 14: La correspondance entre le document LimSee3 et le format intermédiaire 45
Figure 15: Le diagramme de classes d'exportation 46
Figure 16: La représentation de l'ordonnancement de syncbase 50
Figure 17: La représentation d'ordonnancement de (a) container "seq" et (b) container "par" 50
Figure 18: La représentation d'ordonnancement de container "excl" 51
Figure 19: Un exemple de modèle de graphe de ordonnanceur JavaScript 52

1. Introduction

1.1 Contexte

De nos jours, les *applications multimédias* qui servent aux domaines très variés comme l'apprentissage à distance, la télémédecine, les loisirs… sont de plus en plus complexes. Ces applications contiennent des *documents multimédias* qui se composent d'*objets médias* (ex. image, vidéo, audio, hypertexte, animation) obtenus à partir de sources hétérogènes qui se synchronisent les unes avec les autres. La création de ces documents est complexe et délicate. Les problèmes concernant les documents multimédias sont catégorisés en trois types : (1) le format de représentation, (2) l'outil d'édition, et (3) le lecteur du document.

Le format contraint le contenu et le comportement d'un document. En plus, un format standard permet aux auteurs de publier leurs produits en étant sûr que tout lecteur conforme au standard pourra exécuter correctement le document. Il existe des formats dédiés propriétaires comme Flash, MPEG4[1], ou des standards ouverts comme SMIL pour des documents multimédias.

Les outils d'édition de document multimédia permettent de composer des sources médias hétérogènes et de produire des documents publiables sur l'équipement terminal (projecteur, écran, papier,…). De nombreuses approches ont été expérimentées pour offrir des services d'édition de document multimédia [Rabin 1996], [Bulterman 2005], [Deltour 2005]. Normalement, un outil d'édition se base sur un modèle d'édition de document multimédia.

Le lecteur sert à la restitution d'un document multimédia sur l'équipement terminal. Il prend en compte les interactions de l'utilisateur et les traite.

Le travail dans ce stage vise à contribuer au deuxième point de la problématique des documents multimédias, plus précisément, à *la définition d'un modèle d'édition de document multimédia* et à *l'exportation de documents spécifiés dans ce format d'édition vers des documents standards*.

[1] Voir plus informations sur les sections 4.3 et 4.5

1.2 Cadre de travail

Le travail de stage intitulé de « modèle d'édition de document multimédia » s'est déroulé au sein de l'équipe WAM [WAM], INRIA Rhône-Alpes, France pendant 6 mois. Le sujet été proposé dans le contexte du projet LimSee3 [LimSee3] dans le cadre d'une coopération entre l'INRIA et le projet européen Palette [Palette].

L'équipe WAM aborde quelques problèmes posés par les évolutions du Web. Elle se focalise sur la transformation de documents considérée comme un type de traitement générique des documents du Web, particulièrement des documents multimédias.

Le projet LimSee3 vise à développer une nouvelle génération de logiciel source libre pour l'édition de document multimédia utilisant une approche de template. Palette est un projet européen qui vise la facilité et l'augmentation individuel et apprentissage organisationnel pour la coopération dans un groupe d'utilisateurs.

La partie théorique de ce stage comprend l'étude des modèles de document multimédia, l'analyse du modèle d'édition de document multimédia LimSee3 et des propositions d'exportation de document LimSee3 vers des formats standards. La partie pratique propose des expérimentations qui sont intégrées dans le logiciel LimSee3, en particulier, la fonction d'exportation.

1.3 Plan du mémoire

La suite de ce mémoire est organisée en trois parties qui sont les suivantes :

- Première partie : L'analyse du contexte de travail et des problèmes à étudier.
- Seconde partie : L'état de l'art où des solutions technologiques sont présentées en plus des évaluations de ces technologies.
- Troisième partie : La contribution où nous proposons des solutions pour les problèmes d'exportation et de template. Nous présentons aussi des expérimentations et les résultats obtenus.
- La conclusion, les perspectifs et les références terminent le mémoire.

Première partie :

Analyse du sujet

2. Analyse du sujet

Les études dans le cadre de mon stage se divisent en trois parties principales : les études générales sur le modèle d'édition de document multimédia, l'exportation vers des formats standards, et des templates de LimSee3.

Avec l'évolution des applications sur l'Internet, un document électronique est maintenant une composition complexe d'objets médias, en ajoutant une dimension temporelle en plus des dimensions spatiale et logique. Cette évolution est le résultat de nouvelles capacités technologiques des ordinateurs (CPU, mémoire, disques,…), des réseaux (débit, protocoles), et des techniques numériques de traitement des données multimédias : son, vidéo, hypertexte, animation (codage, compression/décompression). Une conséquence est que la création des documents multimédias est de plus en plus complexe du fait qu'un document de ce type intègre par des compositions spatiale et temporelle des médias de types différents à partir de différentes sources (figure 1).

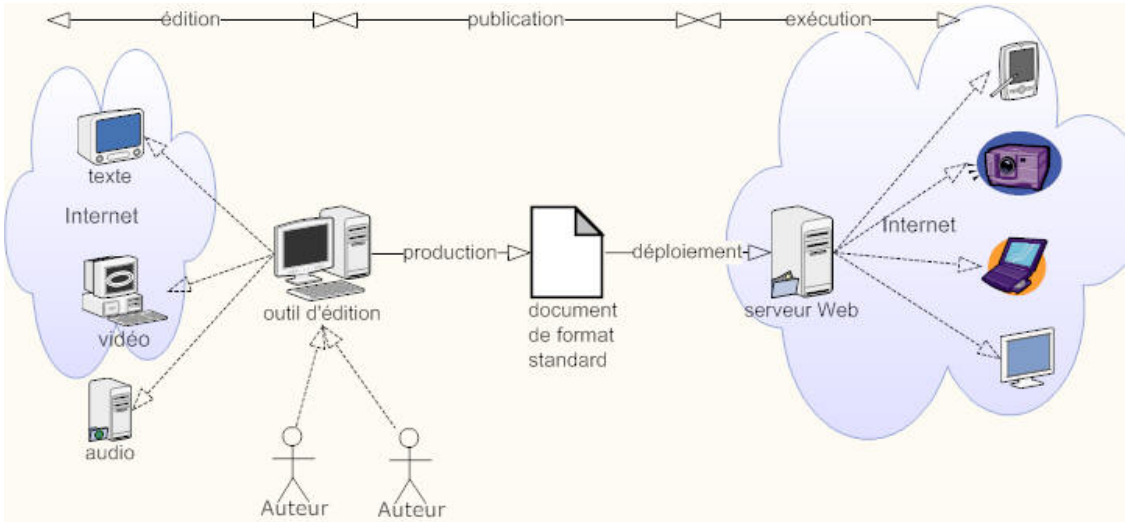

Figure 1: Le processus de création de document multimédia

Le cycle de vie d'un document multimédia passe par trois étapes :

1. L'étape d'édition où les auteurs rassemblent des objets médias, et les organisent en structures logiques, spatiales et scénarios temporels. Les structures de document sont très différentes d'une application à l'autre. Par exemple, un *slideshow* se compose d'une chaîne de transparents qui contiennent un titre, une note de bas de page, et un contenu textuel. Un document de vidéo à la demande se compose de « *trailers* », et de courtes descriptions. On a donc besoin de structures

spécifiques pour différents types d'applications. L'étape d'édition produit des documents sous une forme qui dépend de chaque outil et ainsi un format de document a été définit dans le projet LimSee3.

2. L'étape de publication où les auteurs mettent en place le résultat de l'éditeur sous une forme adaptée. Le document d'édition doit être transformé vers un format standard qui permet d'exécuter sur la plupart de lecteurs.
3. L'étape d'exécution sert à jouer des documents. Le lecteur décode le contenu des documents, résout des références vers des objets médias et restitue le document sur des périphériques terminaux (l'écran, le haut-parleur).

2.1 Type d'application

Les applications multimédias visées par le projet LimSee3 sont variées et utilisées dans des domaines différents. Leurs complexité est ainsi très variable. Par exemple, un album de photos familiales est une application simple permettant d'assembler des photos avec leur titre et leur description. Au contraire, une visite virtuelle d'une exposition est définie par une structure complexe qui contient des textes, des références de vidéos, des audio d'explication. Ces objets médias de sources hétérogènes sont intégrés et synchronisés selon des contraintes temporelles et spatiales.

On peut classer ces différentes applications par leur domaine d'utilisation:

- Éducation et apprentissage : incluant l'instruction de l'aide par ordinateur, l'apprentissage interactif à distance, et l'encyclopédie multimédia.
- Public : incluant la bibliothèque numérique, le musée virtuel.
- Divertissement : incluant la vidéo à la demande, la TV interactive, l'album électronique, le journal électronique personnalisé, la messagerie multimédia.
- Office/commerce : incluant le système de consultation à distance, la visioconférence, le courrier électronique multimédia, la publicité, le travail collaboratif, la publication électronique, le magasin en ligne (médical, bancaire, tourisme…), l'exposition en ligne.
- Autres : incluant le système de surveillance en ligne, le système de sécurité multimédia…

2.2 Format de document multimédia

Comme on peut voir dans la figure 1, lors de l'étape de publication, le document créé doit être exporté vers des formats standards dédiés de la représentation. Le document pourra

être interprété par des lecteurs. Avec le développement d'outils dédiés aux applications multimédias, de nombreux langages sont définis et expérimentés pour la présentation des documents multimédias dont voici les plus représentatifs :

- *Synchronized Multimedia Integration Language* [W3C.SMIL] est un langage recommandé par W3C pour les documents multimédias qui permet d'intégrer des objets médias, des structures spatiales et des scénarios temporels dans un document. Il support aussi des animations, des transitions et même des synchronisations entres des objets médias.
- *Scalable Vector Graphics* [W3C.SVG] : il s'agit en fait d'un format pour les graphiques vectoriels. Sa dernière spécification importe des nombreux concepts de SMIL pour la synchronisation des objets médias et même l'animation.
- *Adobe Flash* [Flash] est un format propriétaire d'Adobe qui permet de contenir des graphiques vectoriels, des objets multimédias, des animations, et des scripts.
- *MPEG4* : est un standard développé par *Moving Picture Experts Group* [MPEG4]. MPEG4 est utilisé sur le Web, pour la conversation (visio-téléphone), pour la diffusion de vidéo sur les chaînes télévisées…

À côté des formats standards, des solutions ad-hoc sont étudiées et expérimentées pour rendre un document multimédia exécutable sur le Web [Schmitz 2000]. En effet, des technologies XHTML avec JavaScript et CSS ont montré leurs capacités de représentation des contenus multimédias, [Raggett 2005], [Meyer 2006]. En plus, ces approches sont supportées par la plupart de navigateurs du Web (ex. Internet Explorer, Firefox, Sarafi, Opera, Amaya).

2.3 Template

Le « *template* » [Mikáč 2006a] est une structure prédéfinie d'un document multimédia qui réduit la complexité de création et sert à la réutilisation. Par exemple, un utilisateur final en profitant des templates créés par des experts peut concevoir sa propre présentation en se concentrant sur les détails et en ajoutant des paramètres avec moins d'efforts.

Le modèle d'édition LimSee3 intègre la possibilité d'utiliser des templates pour permettre d'adapter l'outil d'édition aux différents types d'application. Par contre, il est indépendant des formats cibles.

Un objectif de ce travail est le devoir dans quelle mesure les structures de templates facilitent la transformation vers d'autres formats du fait qu'elles définissent une structure globale de document et sont ciblées vers certains types d'application.

2.4 Synthèse

Les travaux proposés par ce stage requièrent des connaissances sur les langages XML, des modèles de document multimédia, des modèles d'édition de document multimédia. Avec le modèle proposé pour LimSee3, il faut trouver des solutions pour l'exportation vers des documents cibles et en prenant en compte les définitions de templates spécifiques pour certaines applications.

Deuxième partie :

État de l'art

Dans cette partie, nous présentons des technologies concernant le projet LimSee3 et mon stage. Dans le domaine d'édition de document multimédia, il existe des travaux et des expérimentations incluant des projets académiques (*CMIFed* [Rossum 1993], *LimSee2* [LimSee2], *HyperProp* [Soares 2004]) et des produits commerciaux (*Director* [Director], *Authorware* [Authorware],…). LimSee3 favorise les avantages des technologies XML [W3C.XML] et surmonte certains inconvénients d'outils existants. Nous avons donc étudié les langages XML (chapitre 3), des modèles de documents multimédias (chapitre 4), et des modèles d'édition de document multimédia (chapitre 5). L'équipe WAM a proposé un modèle d'édition de document multimédia qui est en cours d'expérimentation et forme le cœur du logiciel LimSee3. Le chapitre 6 introduit et analyse ce modèle. Le chapitre 7 présente l'architecture de LimSee3.

3. Technologie XML

XML est un langage informatique de balisage générique qui est utilisé pour exprimer des langages de balisages spécifiques (exemples : XHTML, SVG, XSLT) par W3C. Son objectif initial est de faciliter l'échange automatique de contenus entre systèmes d'informations hétérogènes, notamment sur Internet. Le document XML possède les propriétés suivantes :

- Permettre de naviguer facilement dans le document.
- La structure d'un document XML est définissable et validable par un schéma.
- Un document XML est entièrement transformable dans un autre document XML et même vers d'autres formats textuels (XHTML, texte).

3.1 Navigation de document XML

Le développement des langages différents basés sur XML conduit le besoin commun de la navigation dans les documents XML. Le XPath [W3C.XPath] est dédié à une portion d'un document XML. Une expression XPath caractéristique est un chemin de localisation constitué par une suite d'éléments ou d'attributs séparés par une barre de fraction «/». XPath fournit des fonctions intégrées, permet d'utiliser des variables et des expressions régulières, et d'évaluer des expressions de calcul. Ce langage est largement intégré dans les langages XSLT [W3C.XSLT] et XQuery [W3C.XQuery]. Il est également utilisé par XPointer [W3C.XPointer] et XLink [W3C.XLink] pour enrichir les liens dans un document XHTML.

3.2 Langages de Schéma

En général, *un langage* XML est défini par sa syntaxe qui décrit des vocabulaires que le langage utilise et les contraintes respectées à son utilisation, et sa sémantique qui ajoute du sens au document. *Un schéma* est une définition formelle de syntaxe d'un langage basé sur XML. *Un langage du schéma* est une définition formelle pour une expression de schéma. Le document utilisant un schéma peut être validé par un processeur qui implémente le langage du schéma.

La technologie XML permet de créer un nouveau langage grâce au langage de schéma. Par exemple, le langage XHTML est contraint et validé par un schéma de DTD. Il existe de nombreux langages de schémas différents : DTD [W3C.DTD], XML Schéma [W3C. XMLSchema], DSD2 [DSD2], RELAX NG [Clark 2003].

Le processus de validation vérifie si un document XML correspond à un schéma. On peut différencier des niveaux de validation :

- La validation de la structure d'un document
- La validation du contenu de nœud individuel : vérifier le type de donnée
- La validation d'intégrité : par exemple, les liens entres des nœuds dans un document ou entre des documents.
- La validation des règles métiers

La validation d'un document XML est coûteuse. Donc, il faut bien choisir le langage de schéma et le niveau de validation. Des nombreuses études sont effectuées sur les langages de schéma, leurs capacités et leurs performances [Lee 2000], [Vlist 2001], [Murata 2000]. Le langage DTD est simple et supporte toutes les structures basiques. Le langage XML schéma supporte en plus des structures, le typage de données et également des expressions de règles métiers.

3.3 Transformation de documents XML

XML fournit un format général pour la représentation de la structure logique des données. Les données peuvent être présentées dans un navigateur ou bien avec d'autres médias (papier, audio, projecteur,…).

XSL transformation est un langage basé sur XML dédié à transformer un document

XML. Il est largement utilisé pour convertir des données entre des schémas différents, transformer document XML au format d'une page Web. En plus, XSLT utilise XPath pour naviguer dans le document XML.

3.4 Synthèse

Avec le développement rapide de technologie, XML devient un standard pour des documents électroniques. Ces technologies accompagnées avec des implémentations complètes qui sont disponibles dans suites logicielles de source libre permettent de valider, naviguer, manipuler, transformer des documents.

4. Modèle de document multimédia

Un modèle de document multimédia est un modèle de document qui sert à décrire des aspects différents d'un document multimédia incluant des structures logique, spatiale, hypertextuelle, et temporelle. Un document est conforme à un modèle s'il satisfait des contraintes définies par ce modèle.

Un modèle dédié pour des documents multimédias doit fournir des caractéristiques suivantes [Thuong 2003], [Boll 2000], [Jourdan 2004]:

- Le modèle dédié pour des documents multimédias est capable d'*intégrer des objets médias différents*. En effet, à l'étape d'édition, des objets médias de sources hétérogènes sont rassemblés en organisant des aspects spatiaux et temporels en plus de la construction des structures logiques.
- Il doit autoriser de *synchroniser les éléments selon la dimension temporelle* qui permet de rajouter le temps dans un document. Par exemple, il fournit des moyens pour spécifier la durée affichée d'une image sur l'écran, ou bien qu'un texte fait suite à une vidéo.
- Le modèle doit permettre d'exprimer qu'un document multimédia *récupérer et traite des événements* générés par le système ou des interactions par des utilisateurs. Par exemple, un clic sur une image transmet vers un lien extérieur.
- Le modèle doit offrir des définitions *d'animations et de transitions* qui rajoutent ainsi de l'attrait à un document.
- Il doit permettre d'exprimer la *sélection de contenu* selon des configurations du système ou du lecteur. Cette capacité rend le document adaptable à des environnements différents (périphérique, débit,…).

- Il doit enfin permettre la définition de *métadonnées* pour permettre aux moteurs de recherche d'indexer les documents selon leurs caractéristiques importantes.

Depuis quelques années, le déploiement et l'utilisation des documents multimédias dans les réseaux amène des recherches sur ce domaine. Le langage XHTML étant un format standard dédié pour la présentation textuelle pose des limitations de présentation des documents complexes. La plupart des solutions actuelles utilisées sur le Web pour permettre l'accès à des données multimédias sont extérieures au standard XHTML. Par exemple, Flash, MPEG4 sont largement utilisés sur l'Internet en rajoutant des nouvelles fonctionnalités aux documents XHTML sous la forme de plugins ajoutés aux navigateurs. Notons cependant que ces formats sont propriétaires et sous la forme binaire. Au contraire, le standard SMIL proposé par W3C est un format ouvert dédié pour la présentation de document multimédia.

Les sections suivantes présentent quelques modèles de documents multimédias existants.

4.1 Synchronized Multimedia Integration Language

Synchronized Multimedia Integration Language (SMIL) [W3C.SMIL] est un langage déclaratif défini et recommandé par le W3C depuis 1998 qui a pour but de permettre l'intégration de contenus multimédias diversifiés (images, sons, textes, vidéo, animations, hypertexte) en les synchronisant afin de permettre la création de présentations multimédias. La structure d'un document SMIL décrit le déroulement temporel et spatial des différents composants intégrés dans une présentation multimédia, sous forme de documents XML. Il permet donc d'indiquer le moment où un contenu sera affiché, pendant combien de temps et dans quelle partie de la fenêtre d'affichage.

Pour faciliter l'utilisation de ce standard dans les contextes très variés (desktop, télévision, téléphone portable,…), deux mécanismes ont été mis en place.

1. La *modularisation* est une approche dans laquelle la fonctionnalité de balisage est spécifiée par un ensemble de modules qui regroupent chacun des sous-ensembles sémantiques par d'éléments XML de SMIL, d'attributs, et de leurs valeurs.
2. Le *profilage* est la création d'un langage basé sur XML en combinant des modules différents afin de fournir les fonctionnalités pour les applications particulières. Dans SMIL 2.1, 50 modules sont regroupés dans 4 profils : *Language profile, Mobile profile, Basic profile, et Extended Mobile profile*.

Il existe des lecteurs capables de lire SMIL (QuickTime, Ambulant, RealPlayer). Quelques navigateurs ont aussi implémenté certains modules de SMIL permettant de lire parti de ce langage au sein d'autres langages. Par exemple, Firefox et Opera peuvent jouer des animations de SMIL. En plus le profil MMS (Multimedia Messaging Service) est un profile de SMIL destiné aux utilisateurs de téléphones mobiles par offrir un service de messagerie multimédia et est supporté par la plupart de réseaux téléphoniques et des téléphones portables.

4.2 Flash

Flash [Flash] est développé et distribué par la société Adobe. Il comprend un logiciel utilisé pour créer le contenu du Flash, et un logiciel lecteur de ce format qui est une application client fonctionnant sur la plupart des navigateurs. Flash permet de définir des graphiques vectoriels et des bitmap, et de définir des comportements, des animations à l'aide d'un langage de script appelé *ActionScript*. Enfin, il permet de gérer selon un flux bidirectionnel de l'audio et de la vidéo. Le format de fichier Flash est binaire.

Depuis son lancement en 1996, la technologie Flash est devenue une des méthodes les plus populaires pour ajouter des animations et des objets interactifs à une page Web. De nombreux logiciels de création et des systèmes d'exploitation sont capables d'afficher du Flash. Flash est généralement utilisé pour créer des animations, des publicités, des jeux vidéo, ou des applications riches. En effet, il permet aussi d'intégrer de la vidéo en streaming dans une page, jusqu'au développement d'applications multimédias comme des lecteurs multimédias, ou des logiciels de traitement de texte en ligne.

Les lecteurs de Flash sont disponibles sur la plupart des navigateurs (comme Internet Explorer, Firefox, Safari, Opera, Konqueror,…) sous la forme de plugin.

4.3 MPEG-4

MPEG-4 [MPEG4] est d'abord conçu pour gérer le contenu de scènes comprenant un ou plusieurs objets audiovisuels. Les usages de MPEG-4 englobent toutes les nouvelles applications multimédias comme le téléchargement et le streaming sur Internet, le multimédia sur mobile, la radio numérique, les jeux vidéo, la télévision et les supports haute définition.

Les groupes de travail de MPEG-4 ont aussi développé de nouveaux codecs audio et

vidéo et ont permis l'enrichissement de contenus multimédias, en ajoutant de nouvelles fonctions comme l'hypertexte, le support pour des présentations 3D, des fichiers composites (incluant des objets audio, vidéo, animation et hypertexte), le support pour la gestion des droits numériques et plusieurs types d'interactivité.

Le standard MPEG-4 se décompose en une suite de normes, les parties, qui spécifient chacune un type de codage particulier. Dans chaque partie, plusieurs profils (collection d'algorithmes) et niveaux (contraintes quantitatives) sont définis. Un consortium industriel désirant utiliser MPEG-4 choisit une ou plusieurs parties de la norme et, pour chaque partie, il peut sélectionner un ou plusieurs profils et niveaux correspondant à ses besoins.

MPEG-4 est utilisé dans des applications variées comme la télévision numérique, des animations graphiques, des jeux vidéo, des vidéos en streaming… De nombreux lecteurs implémentent MPEG-4 au plutôt conformes des sous-ensembles à certaines parties, comme DivX, Xvid, Nero Digital, Quicktime.

4.4 Scalable Vector Graphics

Scalable Vector Graphics [W3C.SVG] est une spécification d'image vectorielle proposée par W3C. Les coordonnées, dimensions et structures des objets vectoriels sont indiqués sous forme numérique dans le document XML. Un système spécifique de style (CSS ou XSL) permet d'indiquer la décoration et les polices d'écriture à utiliser.

Ce format gère quelques formes géométriques de base (rectangles, ellipses, etc.), mais aussi des chemins, qui utilisent les courbes de Bézier et permettent ainsi d'obtenir n'importe quelle forme. Le remplissage peut se faire à l'aide de dégradés (gradients) de couleurs de motifs qui sont des objets SVG quelconques, ou de filtres. On peut également appliquer des motifs le long des chemins et utiliser les fonctions de remplissage. Le canal alpha, pour la transparence, est géré à tous les niveaux.

Comme dans tout document XML, les objets sont organisés sous forme d'arbre. Le format permet l'intégration d'animations, ou la manipulation des objets graphiques par programmation, notamment grâce à des scripts qui peuvent être intégrés dans SVG.

Un des intérêts majeurs de SVG est qu'il peut être inclus dans d'autres documents XML, comme par exemple des documents XHTML ou des documents XML devant être traités

par des langages de transformation. Respectant la norme XML, une image SVG peut également être manipulée par l'intermédiaire du modèle Document Object Model (DOM).

SVG peut être visualisé nativement avec certains navigateurs Web, comme Konqueror, Opera, et Mozilla Firefox, ou à l'aide d'un plugin pour d'autres.

Sur le plan multimédia, ce qui nous intéresse est la façon de faire du multimédia avec SVG donc l'intégration du temps pour les animations, les transitions, l'intégration de médias temporels.

4.5 Synthèse

En reprenant les critères définis dans l'introduction de ce chapitre, nous pouvons comparer les différents langages multimédias selon le tableau ci-dessous :

LimSee3	SMIL	Flash	MPEG4	SVG
Intégration	x	x	x	partiel
Synchronisation	x	x	x	partiel
Événement	x	x	x	x
Animation	x	x	x	x
Transition	x	x	x	x
Sélection de contenu	x	x	x	x
Méta donnée	x			x
Contenu	référence	fichier unique	fichier unique	référence
Format	XML	binaire	binaire	XML
Licence	libre/ W3C	propriétaire/ Adobe	propriétaire/ ISO/IEC	libre/ W3C
Lecteur	lecteur SMIL(1)	plugin de navigateur	lecteur de MPEG4	La plupart de navigateurs(2)

(x) : totalement supporté.

(1): AMBULANT, Helix Player, QuickTime Player, RealPlayer, et des plugins de navigateurs,...

(2): Firefox, Safari, Opera, Konqueror, Amaya.

Ces solutions fournissent des fonctionnalités assez complètes, sauf que les navigateurs modernes ne supportent pas entièrement chacune de ces technologies. Avec l'usage du document multimédia augmenté par des services multimédias, comme le partage de vidéo, le besoin de déploiement plus large de ces produits sur les navigateurs devient de plus en plus important.

Des nombreuses approches ad-hoc sont étudiées et expérimentés pour rendre des services multimédias accessibles depuis un navigateur du Web. Par exemple, S5 [Meyer 2006], Slidy [Raggett 2005] pour des transparents. Dans ces outils, le navigateur appelle des lecteurs de média (audio, vidéo, animation) qui lancent des plugins pour la présentation d'objets médias. La synchronisation est cependant difficile car le navigateur n'autorise pas JavaScript à récupérer et à intervenir sur les événements générés par des plugins. D'autre part, le navigateur lui-même pose des limitations car il ne supporte pas des fonctions comme les transitions, les liens temporels, les animations.

Les limites et les contraintes évoquées ci-dessus seront à prendre en compte lors de notre analyse du problème de l'exportation depuis LimSee3 vers ces formats. En particulier, nous prenons en compte leurs diversités dans notre architecture d'exportation.

5. Modèle d'édition de documents multimédia

De nombreuses approches ont été étudiées et expérimentées pour offrir des services d'édition de documents multimédias. Ces outils répondent de façon variée aux critères de base qui sont la facilité de création et le pouvoir d'expression des différentes fonctions des documents multimédias. Les outils dédiés aux des langages standards comme SMIL (GRiNS [GRiNS], LimSee2 [LimSee2]) proposent aux utilisateurs de manipuler directement la structure du langage. Ils demandent cependant aux auteurs de bonnes connaissances sur les standards. Ils sont donc difficilement utilisables pour la plupart des concepteurs qui n'ont pas les compétences pour maîtriser ces langages techniques. Les outils commerciaux (PowerPoint [PowerPoint], Director [Director]) sont limités en fonctionnalités mais plus flexibles et simples pour les utilisateurs. Comme les modèles d'édition de document multimédia restent complexes, aucun outil ne fournit une solution complètement satisfaisante.

L'édition de document multimédia est un processus complexe qui demande aux auteurs de spécifier des informations de type différent en niveau différent. Ces tâches incluent la sélection de sources des objets média qui seront présentés, les agencements spatiaux, les liens/les relations d'interaction, et les relations temporelles entre eux. La transmission de documents qui sont accédés via le réseau conduit à une complexité additionnelle pour spécifier le contenu alternatif pour adapter la présentation à l'environnement de restitution (la bande passante, ou la capacité de l'appareil).

5.1 Fonctions d'édition

Plusieurs systèmes ont abordé la complexité du processus d'édition multimédia selon des approches différentes. La plupart des systèmes commerciaux réduisent la complexité d'édition en limitant les capacités de présentation. Plusieurs outils issus de la recherche essaient de fournir des supports compréhensifs pour créer des présentations complexes.

Lien avec les critères de « modèle de document multimédia », un modèle d'édition des documents multimédias comprend les caractéristiques suivantes [Bulterman 2005], [Deemter 2000], [Jourdan 2001]:

- Ensemble de médias : Ce sont les objets médias qui sont accédés par références vers des ressources dans le réseau. Typiquement, un exposé comprend un titre, des images extraites dans plusieurs sources sur l'Internet, des audio commentaires créés par l'auteur (ressource locale) ou référencés. Certaines propriétés d'objets médias peuvent être inconnues au moment de l'édition. Par exemple, la durée d'un *newscast* n'est pas explicite. L'outil d'édition donc doit permettre d'exprimer un média qui a une durée inconnue.
- Composition de synchronisation : Une collection d'objets peut être construite et groupée pour permettre de les représenter et les synchroniser. Ces objets dépendent soit d'une contrainte explicite, soit d'une relation avec d'autres objets (contrainte relative).
- Disposition spatiale : La représentation de plus d'un objet dans une composition en même temps. Les aspects spatiaux sont explicites (les images), dynamiques ou dépendants du temps (les animations, les vidéos). On a besoin d'un mécanisme flexible de manipulation et de réédition.
- Événements asynchrones : les événements peuvent être générés de façon asynchrone pendant la présentation, comme cliquer sur le bouton « suivant ».
- Contenu de remplacement : l'auteur ne connaît pas les paramètres de lecteur au moment d'édition. Par exemple, le débit peut être différent entre des clients. Dans le cas ou le lecteur ne dispose que d'une connexion par modem avec le maximum débit de 56Kbits/s, l'auteur est capable de spécifier au lecteur de télécharger une série d'images au lieu d'une vidéo complète. Le système permet d'identifier des conditions d'exécution et les associer à des contenus à l'édition. Ces paramètres visent le débit, la taille d'écran, la résolution d'écran, la capacité de CPU, le système d'exploitation [SMIL]…

- Réutilisation : les outils permettent de bénéficier des structures prédéfinies, des documents existants. Elle réduit le temps de création d'un document grâce aux structures communes.
- Performance : incluant la complexité de la manipulation de contenu, de l'interaction d'utilisateur.

Les modèles étudiés ci-dessous se basent sur des critères ci-dessus.

5.2 Typologie des systèmes d'édition

Un outil d'édition de document multimédia est un système complexe qui doit permettre d'offrir à l'auteur le moyen de manipuler les aspects différents ensembles de document (logique, spatial, temporel). En général, le modèle d'édition de documents multimédias peut être réalisé selon quatre approches dominantes [Bulterman 2005] : structure, timeline, graphe, ou script.

5.2.1 Approche par modèle de structure

Le modèle par structure s'appuie sur une structure hiérarchique temporelle de document en utilisant la représentation abstraite pour définir l'ensemble des médias et de leurs attributs. Il se compose des nœuds qui contiennent une composition consécutive ou parallèle des objets médias. La composition facilite les manipulations spatiales ou temporelles sur un groupe d'objets. En plus la structure d'arbre permet de déterminer des événements, des liens, et des comportements des objets médias (Figure 2).

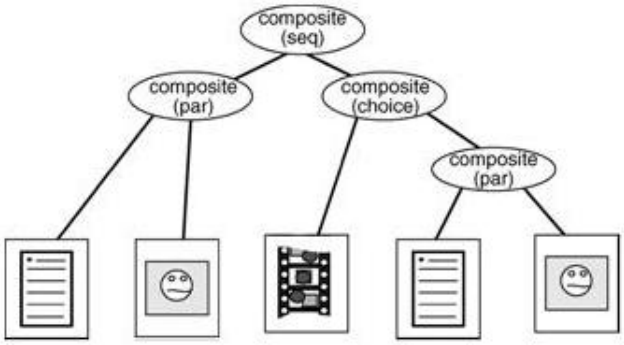

Figure 2: Une structure d'arbre temporel de document (Bulterman 2005)

Madeus [Opéra 2002] est un exemple de modèle d'édition qui étend sur cette approche aux différentes structures (logique, spatiale, temporelle) de document et qui a été développé par équipe Opéra, INRIA. La structure logique d'un document permet de le

décomposer en sous-parties. Les dimensions temporelles et spatiales sont orthogonales. Cette façon de concevoir l'organisation d'un document permet de mieux choisir les langages de spécification et les techniques de formatage associées à chaque dimension. Elle permet aussi de profiter de la structure logique pour déduire des informations qui dépendent des deux autres dimensions, facilitant ainsi la tâche de l'auteur. En plus elle rajoute une vue temporelle.

L'avantage de cette approche est la facilité d'utilisation que permet d'éditeur structuré. Les manipulations sur cela structure permettent de spécifier directement des dispositions et des scénarios temporels d'objets individuels ou groupés. Le contenu de replacement peut être défini par des branches. En plus, les structures communes dédiées à chaque type d'application aident les auteurs à moduler leurs documents et à réutiliser leurs produits. La performance peut être réduire dans le cas de gros document car la manipulation sur des arbres est coûteuse.

5.2.2 Approche par modèle de timeline

Le modèle de timeline s'appuie sur l'axe temporel comme lieu de manipulation des relations logiques entre des objets médias. Ce modèle convient naturellement aux applications comme celles utilisant une liste de lecture (« *playlist* »), la bande de vidéo où l'ordre temporel est important. Chaque objet est manipulé indépendamment des autres plutôt qu'un ensemble d'objets médias (Figure 3).

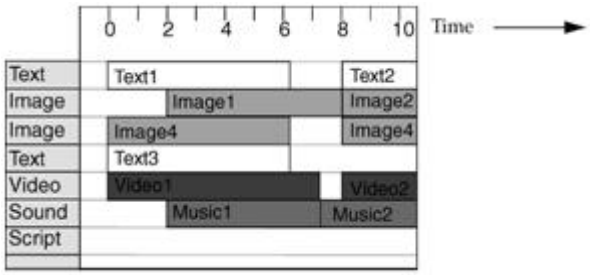

Figure 3: Le timeline d'un document (Bulterman 2005)

Director [Director] est un exemple de modèle de timeline qui permet d'intégrer des objets médias différents pour des contenus d'animation qui sont déployés sur CD, DVD, et aussi sur l'Internet. L'outil expose une vue de temps qui permet aux auteurs de placer directement des objets médias. En plus, la disposition est explicite via la position spatiale dans chaque cadre qui est divisé discrètement en intervalle de temps. Les objets médias sont placés de façon asynchrones dans des cadres.

Cette approche s'adapte bien aux applications où la dimension temporelle est primordiale et facilite des synchronisations entre des éléments médias. Elle manque de structure flexible pour manipuler la disposition spatiale et rend difficile la réédition des documents car les mises à jour peuvent être complexes et fastidieuses.

5.2.3 Approche par modèle de graphe

Le modèle de graphe utilise des graphes pour caractériser et représenter l'intégration et la synchronisation entre des ensembles d'objets médias. Un graphe donne aux auteurs une représentation visuelle des relations complexes entre des objets et également des séquences logiques ordonnées temporellement (Figure 4).

Firefly [Buchanan 2005] est une spécification de comportement temporel pour un document. Il combine un langage contraint avec des notions de temps qui sont modélisées sous forme de graphe. L'avantage de Firefly est l'existence d'un cadre pour exprimer des relations complexes entre des objets.

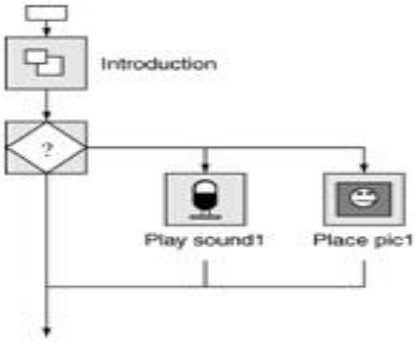

Figure 4: Un document basé sur le graphe (Bulterman 2005)

Cette approche représente tous les aspects différents dans un seul document multimédia. Les relations entre des éléments sont bien définies et manipulées. Les comportements et scénarios du document sont faciles à créer et gérer. Il s'adapte aux applications qui contiennent des nombreux objets médias et des relations complexes. Cette flexibilité conduit les auteurs à des complexités non nécessaires. En plus, les synchronisations sont quelquefois difficiles à exprimer sur un graphe complexe.

5.2.4 Approche par modèle de script

Le modèle de script fournit aux auteurs des langages qui permettent de programmer des positions et des timings d'objet individuel, et même de récupérer des événements. Il

fournit un moyen de contrôler la synchronisation et la désynchronisation des éléments médias et des scénarios du document.

Nsync [Bailey 1998] est une approche utilisant un langage script qui permet de gérer des contraintes temporelle et spatiale. Ces contraintes sont déterminées par des variables. Il utilise des événements pour activer des sous-structures dans la présentation. Ce mécanisme rend l'ordonnancement efficace en temps d'exécution. Cependant, la structure de document et la maintenance sont source de problèmes pour les auteurs.

Cette approche rend possible la création des scénarios complexes de documents multimédias. Elle fournit un mécanisme très bas-niveau. Sur le plan du support pour l'édition, il manque la représentation de document dans les axes spatial et même temporel. Donc, il n'est adapté qu'aux auteurs experts.

5.3 Synthèse

L'édition de document multimédia joue un rôle important et devient de plus en plus complexe. Il demande des caractéristiques communes pour des logiciels et même dédiés pour l'outil d'édition. Les modèles variés ont été étudiés et expérimentés ; ils visent à rendre les tâches d'édition plus faciles aux auteurs. En effet, le modèle de timeline en fournissant une vue temporelle sera bienvenu pour un document plutôt orienté temps et simple. Le modèle de script supporte largement des scénarios et des comportements complexes de document. Il manque de simplicité d'utilisation et demande de bonnes connaissances aux auteurs. Les modèles de structure et de graphe fournissent des relations virtuelles entre des éléments multimédias. Par contre, un changement de scénario peut détruire ces structures et devenir complexe.

Il n'existe pas de meilleure solution pour tous les besoins d'édition. Les applications multimédias diversifiées conduisent à des approches différentes.

Dans certains outils d'édition, plusieurs modèles d'édition sont utilisés autour d'un modèle principal. Par exemple, LimSee2 [LimSee2] étant un éditeur pour SMIL se base sur la structure logique de modèle SMIL en rajoutant des vues temporelle et spatiale que les auteurs peuvent manipuler directement. Ce mélange rend l'outil efficace et adapté aux différents types d'éditeur.

6. Modèle de Document LimSee3

Les formats standards pour des documents multimédias sont dédiés pour la présentation des objets médias dans les lecteurs. Par conséquent, ils sont difficiles à manipuler et à utiliser dans l'étape d'édition. La manipulation directe de langage comme SMIL est trop complexe pour la plupart d'utilisateurs car ces tâches demandent des bonnes connaissances de la sémantique du langage. En effet, un auteur doit comprendre la structure temporelle et les sémantiques d'événement pour créer des relations temporelles entre des objets médias.

LimSee3 vise à répondre plus efficacement aux objectifs d'édition multimédia. L'équipe WAM a récemment proposé une nouveau modèle pour l'édition de document multimédia [Deltour 2006a], [Deltour 2006b]. Il intègre non seulement des mécanismes d'objet permettant la structuration à la fois logique, spatiale et temporelle tout en favorisant la modularité et la réutilisation, mais également des éléments de construction de templates. Ce modèle est un langage métier de XML et s'appuie sur une DTD [Mikáč 2006a], [Deltour 2006a] qui définit des contraintes de contenu du document. Donc l'approche favorise l'utilisation des standards ouverts et des techniques libres comme XPath, XSLT.

6.1 Structure

Le modèle se base sur l'utilisation d'un mécanisme de *template* (6.2) qui est contraint par un schéma DTD avec son espace de nom. Les documents liés par ce modèle sont capables de générer les documents finals en format standard grâce au processus de transformation. Cette approche basée sur des templates définit la dimension logique comme la structure principale de document qui permet de construire un arbre de composants modulaires étant contrainte par un mécanisme de template dédié.

Le modèle (figure 5) se compose deux branches principales : (1) l'en-tête définit des métadonnées pour un document comme le titre, le créateur, le date de création…. (2) le corps comporte des définitions logique, spatiale, et temporelle et des définitions de templates d'un document. Dans ce contexte, l'objet contient des éléments et des attributs permettant de décrire complètement tous les aspects d'un ensemble d'objets médias.

Ce modèle est fortement relié à celui de SMIL en utilisant des concepts de SMIL comme la région, des containeurs du temps (parallèle, séquence, exclusif) et ses attributs. En plus, il rajoute de nouveaux concepts : (1) les références (temps, disposition, objet,

modèle) qui permettent de référer vers un objet existant local ou dans autre document ; (2) *object* permet de définir un ensemble d'objets et ses comportements ; (3) *timing, layout, children* permettent de structurer le temps, disposition, et l'ensemble d'objets contenus respectivement ; (4) modèle permet de définir un template.

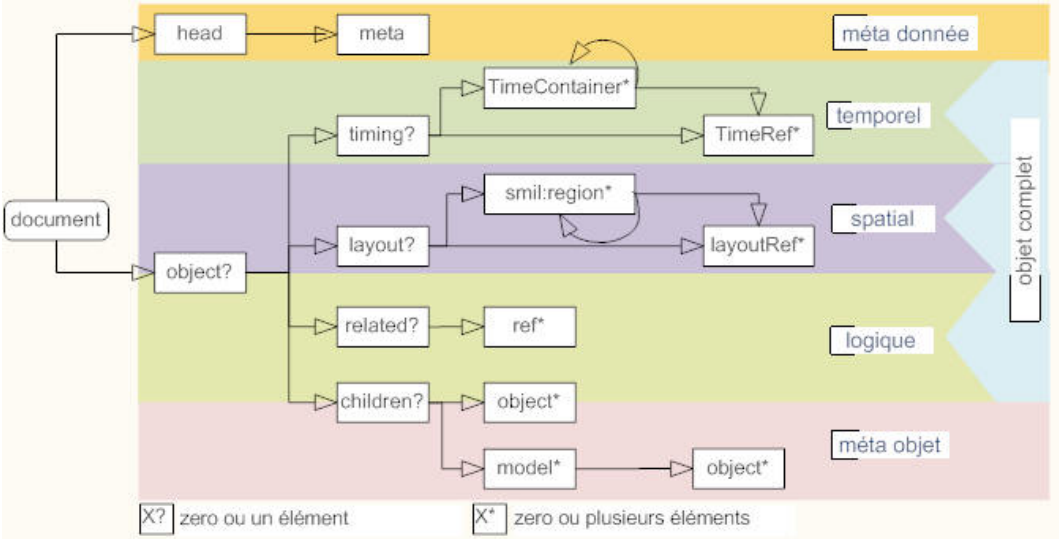

Figure 5: Le schéma du modèle de document LimSee3

L'objet complexe se compose des objets, des instances de template et ses définitions spatiale et temporelle et ses descendants. C'est-à-dire qu'un objet complexe décrit tous les aspects du document multimédia incluant les parties logique, spatial, et temporel. En plus, il est désigné par son identité locale qui lui permet d'être attaché dans l'arbre de ses ascendants.

6.2 Template

LimSee3 s'appuie sur un mécanisme de templates qui permettent à l'utilisateur de créer ou de modifier des documents complexes avec un minimum d'effort. Le template de LimSee3 peut être vu comme un « document à trous » : certaines parties sont préétablies et l'on n'a pas à s'en soucier, d'autres attendent de recevoir le contenu choisi par l'utilisateur, qui est alors guidé dans sa démarche par l'application. L'intérêt principal de ce mécanisme est la rapidité de la mise en œuvre : même un utilisateur débutant est rapidement capable de créer des documents de qualité, à condition bien sûr que le modèle correspondant à ses besoins soit disponible. Il supporte un processus de l'édition progressive de documents multimédia [Mikáč 2006b].

Les utilisateurs expérimentés peuvent adapter les modèles existants à leurs besoins particuliers, ils peuvent en créer de nouveaux et ils peuvent même ne pas utiliser de modèle du tout : il reste possible de concevoir les documents de A à Z, sans contrainte d'un modèle. C'est à l'utilisateur que revient de choisir entre la flexibilité totale de l'édition libre et la simplicité d'utilisation d'un modèle, sachant que tous les degrés intermédiaires entre la liberté et la contrainte extrêmes sont possibles [Mikáč 2006b].

Deux types de composants de templates sont définis : place-holder et modèle complexe :

Place-holders

Un place-holder est un modèle qui a pour but de définir un emplacement pour un objet média. Ce modèle est un objet qui ne spécifie pas la source de média. Cependant, il peut prédéfinir les types acceptables par l'attribut *template:types*. Les noms des types autorisés sont séparés par un point-virgule (Table 1).

Un modèle d'un bouton est défini ci-dessous :

```
<object localId="nextButton" template:types="image ; text" />
```

Quand une instance de ce modèle est créée, l'attribut *src* sera spécifié :

```
<object localId="nextButton" template:types="image ; text"
src="next.jpg" type="image"/>
```

Table 1: Un exemple de template "place-holders" avec une instance

Modèle complexe

Ce modèle plus général est décrit par des objets complexes embarqués par le *template:model* (Table 2). Il est identifié par son nom et permet de créer des instances. Le nombre d'instances minimum et maximum peut être spécifié par les attributs *min* et *max*.

Un modèle d'un bouton est défini ci-dessous :

```
<template:model name="button" min="1" max="4">
  <object template:types="image ; text" />
</template:model>
```

Quand des instances de ce modèle sont créées :

```
<object template:types="image ; video" localId="ok" type="image"
src="medias/ok.png" modelRef="button"/>
<object template:types="image ; video" localId="cancel" type="image"
src="medias/cancel.png" modelRef="button"/>
```

Table 2: Un exemple de modèle complexe

6.3 Caractéristiques

LimSee3 décrit des relations qui existent entre des différents médias, mais les médias eux-mêmes ne font pas partie du document, ils sont référencés par des liens vers des sources médias. Cette approche, tirée du langage SMIL, où un document fédérateur décrit les liens logiques, spatiaux et temporels entre diverses ressources indépendantes présente plusieurs avantages [Mikáč 2006a], [Mikáč 2006b], [Deltour 2006] :

- Les ressources (images, vidéos, animations, textes, pistes audio,...) ne sont pas enfouies dans un fichier opaque, et restent donc facilement réutilisables,
- Les ressources ne sont pas obligatoirement locales : certaines peuvent être situées sur des sites distants et accédées via l'Internet.
- Les documents produits sont de petite taille : le contenu textuel est très compact. Au moment de la présentation, des objets médias peuvent être téléchargés par le lecteur.
- Le contenu textuel rend le document accessible par des moteurs de recherche.

LimSee3 s'attache à adopter le point de vue de l'utilisateur. C'est pourquoi les documents manipulés reflètent la logique de la présentation multimédia plutôt que ses besoins techniques : par exemple, l'enregistrement d'une conférence pourra être organisé en chapitres correspondant à chaque orateur, les chapitres étant subdivisés en sections suivant les différents transparents utilisés. Les médias utilisés viendront ensuite s'insérer dans cette structure logique.

LimSee3 s'appuie sur un mécanisme de modèles qui permettent à l'utilisateur de créer ou de modifier des documents complexes avec un minimum d'efforts.

Enfin, le template est un document de squelette réutilisable qui est le point de départ pour créer une instance. L’utilisateur de template va remplir ses attributs de façon incrémentale. Il s’adapte à la coopération en groupe qui permet à plus d’un auteur de travailler sur un même document.

6.4 Exemple

Dans cette section, nous présentons un exemple qui servira comme base d’illustration du service d’exportation que nous décrivons dans la section 9. Cet exemple représente un transparent simple qui est défini à l’aide d’un template nommé *scene* (ligne 4-6) qui peut

contenir une image ou un vidéo, une vidéo (ligne 7) qui utilise le template *scene*, un titre (ligne 8-10), qui est une instance de template *scene* et un objet textuel (ligne 11) qui décrit le vidéo. Les lignes 13-18 décrivent la structure temporelle du document. Les lignes 19-29 décrivent la structure spatiale du document. L'ouverture et la fermeture de document sont présentées sur les lignes 1 et 31. Ligne 2 décrit un objet global qui contient un enfant (ligne 3-12), la structure temporelle, et la structure spatiale. Notons que l'élément *title* ne définit pas explicitement une référence de temps, il appartient au container de temps défaut d'objet *slide*.

Liste 1: un document LimSee3 simple

```
1. <document>
2.   <object localId="slide" type="complex">
3.      <children>
4.        <template:model name="scene" min="1">
5.          <object template:types="image ; video"/>
6.        </template:model>
7.        <object template:types="image ; video" localId="scene1"
   type="video" src="medias/scene1.jpeg" modelRef="scene"/>
8.     <object localId="title" type="text" src="medias/title.txt" >
9.          <timing dur="indefinite" />
10.          </object>
11.       <object localId="desc" type="text" src="medias/desc.txt"
   backGroundColor="green" showBackground="always" />
12.     </children>
13.     <timing>
14.       <smil:par begin="3s" dur="5s">
15.         <timingRef refId="scene1" />
16.         <timingRef refId="desc" />
17.       </smil:par>
18.     </timing>
19.     <layout width="500" height="300" >
20.         <smil:region regionName="title" left="auto" top="auto"
   width="50" heigh="100%">
21.            <layoutRef ref="title" />
22.         </smil:region>
23.         <smil:region regionName="image" left="auto" top="50"
   width="350" heigh="250">
24.            <layoutRef ref="scene1" />
25.         </smil:region>
26.         <smil:region regionName="desc" left="350" top="50"
   width="150" heigh="250">
27.            <layoutRef ref="desc" />
```

```
28.         </smil:region>
29.       </layout>
30.     </object></document>
```

La figure 6 représente le document ci-dessus par l'axe temporel :

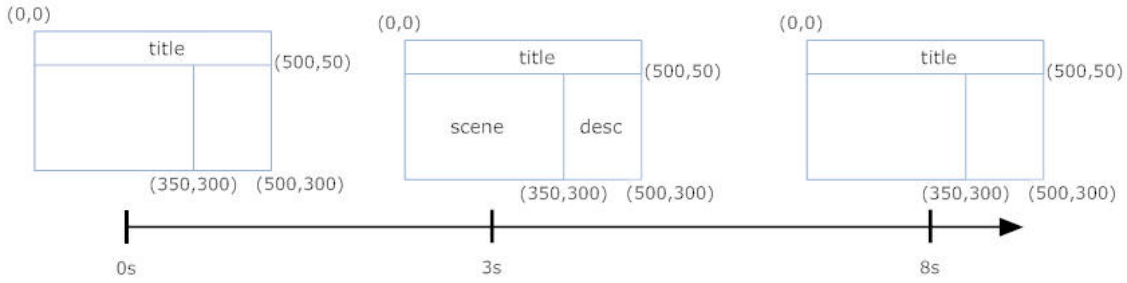

Figure 6: La représentation de document par l'axe temporel

L'élément *title* est activé à l'activation de document car il utilise le container de temps d'objet *slide* qui a l'attribut *start* par défaut. Il affiche en tous temps car sa durée activée est *indefinite*. Les éléments *scene* et *desc* affichent à 3 seconds après l'activation de document car ils appartiennent au container parallèle qui définit le *start=3s*. Ils disparaissent à 8 seconds après l'activation de document car la durée de son container *dur=5s*.

6.5 Conclusion

Le modèle de document LimSee3 dédié pour l'édition de document multimédia s'appuie fortement sur l'utilisation de templates, en rajoutant des macrostructures logique, spatiale, et temporelle. Il rend possible la manipulation d'un document multimédia selon des aspects différents.

Le mécanisme de template fournit aux utilisateurs de la modularisation et la réutilisation de document. Il permet d'être utilisé par des auteurs de niveau varié : les experts peuvent créer des templates complexes et des structures globales d'un document ; les débutants réutilisent ces templates et composent de nouveaux documents.

7. Architecture de LimSee3

LimSee3 est une application d'édition offerte aux utilisateurs. Elle propose une interface évolutive et adaptable qui permet de refléter des besoins particuliers.

LimSee3 [LimSee3] s'appuie sur l'architecture Modèle-Vue-Contrôleur (MVC) qui

divise bien l'application en plusieurs modules. Le modèle se compose de DOM (Document Object Model) LimSee3 et des processeurs qui interprètent le DOM sur les dimensions spatiale et temporelle. La vue contient des classes GUI. Le contrôleur s'appuie sur le motif de conception observateur/observable.

Il fournit des vues multiples qui permettent aux utilisateurs de manipuler les aspects différents d'un document multimédia. En effet, la vue d'objets représente la structure hiérarchique d'objets et permet ainsi d'interpréter facilement les éléments et leurs attributs. Cette structure correspond exactement à la structure du document source LimSee3.

La vue de disposition représente les relations spatiales entre objets. Un mécanisme de manipulation sur la dimension spatiale construit un arbre interne de LimSee3 qui présente la structure hiérarchique spatiale. Le *layoutRef* qui définit la relation spatiale entre avec des objets média est résolu par une table de références. Ainsi, la structure *SpaceHierarchy* représente entièrement les relations spatiales sous forme d'un arbre.

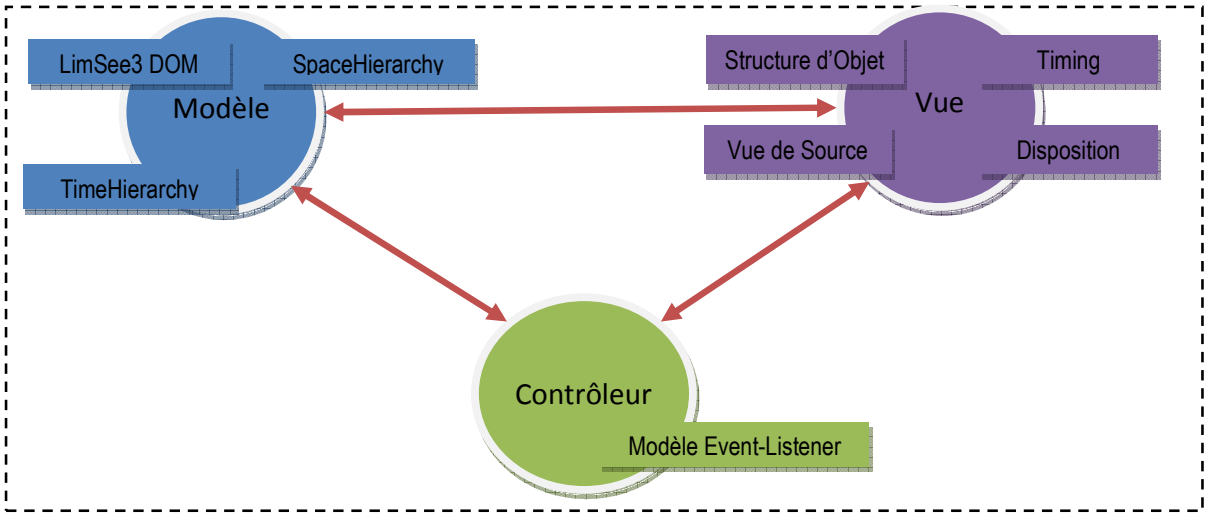

Figure 7: L'architecture de LimSee3

La vue de timing visualise l'organisation temporelle des objets. LimSee3 facilite la manipulation des structures temporelles en laissant aux utilisateurs la possibilité de changer les attributs du temps d'un objet de façon cliquer-et-tirer. Un processeur est construit en structurant le temps par un arbre hiérarchique. Le *timingRef* qui définit la relation entre les éléments temporels avec des objets média est résolu par une table de références. *TimeHierarchy* représente entièrement la relation temporelle sous forme d'arbre.

Troisième partie :

Contribution

Dans cette partie, nous présentons les problèmes liés d'exportation de documents LimSee3 vers des formats de présentation et proposons des solutions incluant la description des expérimentations effectuées ainsi que la prise en compte de templates lors de l'exportation.

8. Exportation

8.1 Introduction

L'exportation est un processus pour convertir un document LimSee3 vers des formats standards de présentation multimédia. Comme on l'a vu, le modèle de document LimSee3 a été défini dans le but de faciliter la création d'un document multimédia. En effet, un template de LimSee3 est un document « squelette » qui aide l'auteur lors du processus de composition d'un document. Ce modèle d'édition nécessite une étape supplémentaire qui consiste la conversion du document créé vers un format cible de présentation. Précisément, l'exportation est la transformation d'un document LimSee3 vers un format de présentation de document multimédia.

Il existe de nombreux formats de document multimédia (voir la section 4) : les formats de présentation basés sur une syntaxe textuelle XML (SMIL, XHTML+JavaScript+CSS, SVG), les formats binaires (Flash, MPEG-4, PowerPoint). Cette diversité rend le problème d'exportation plus complexe.

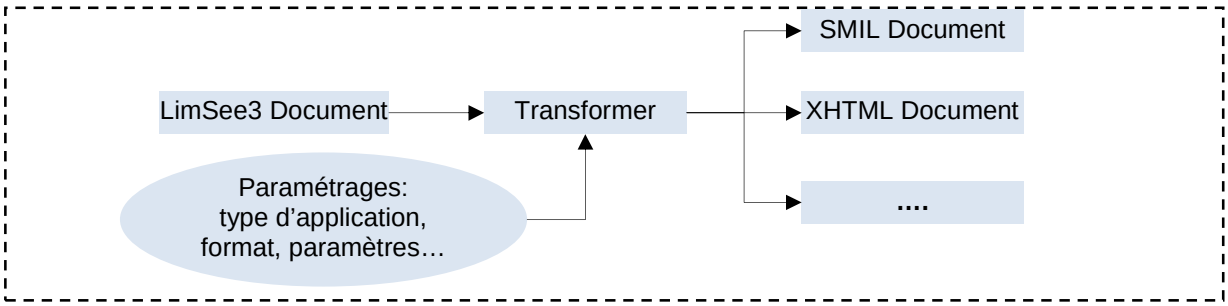

Figure 8: Le processus d'exportation

La difficulté de l'exportation vient aussi de la complexité des documents incluant les structures logique, spatiale et temporelle que les formats cibles prennent en compte de façon très différente. Ainsi SMIL étant spécifié pour des documents multimédias exprime bien les aspects temporels mais ne permet pas d'exprimer de façon simple les structures logiques. Au contraire, XHTML étant dédié à l'hypertexte ne supporte l'intégration d'objets médias qu'au travers des extensions (objets embarqués) et ne permet pas

directement la synchronisation. Il faut donc utiliser JavaScript pour récupérer des événements et faire certaines synchronisations. De plus, comme on l'a vu dans la section 2.1, dans certaines applications, par exemple, des albums de photos, des albums de musique, l'axe temporel domine la structure logique. D'autres sont dominées par l'axe spatial comme la publicité. Enfin, d'autres applications sont le résultat de composition complexe entre la structure logique, la disposition spatiale et le temps, comme dans le cas d'un musée virtuel ou d'un *slideshow* de cours interactif à distance.

La figure 8 décrit le processus d'exportation. Il prend comme entrée un document LimSee3 et le type d'application visée pour permettre d'optimiser le résultat obtenu. La sortie est un document cible de type SMIL, XHTML+JavaScript, Flash, ou MPEG4,…

Le principal travail qui a été mené pendant ce stage a porté sur l'étude, la conception et l'implémentation d'un service d'exportation pour LimSee3. Ce service doit tirer parti de l'architecture et des services existants dans le logiciel existant. En particulier, le cœur de l'implémentation de LimSee3 effectue l'interprétation de la syntaxe de document LimSee3 constituée de fonctions internes (analyseur syntaxique, calcul de dépendances temporelles et spatiales,…) qui fournissent un mécanisme de manipulation des structures de document nécessaires pour ce service d'exportation que l'on souhaite. Ces éléments sont pris en compte à l'entrée du programme d'exportation.

L'organisation de cette section se divise en deux parties : (8.2) l'étude de l'exportation de LimSee3 vers SMIL et vers XHTML, et (8.3) les approches possibles pour implémenter l'exportation.

8.2 Étude de l'exportation de LimSee3

Dans cette section, nous présentons les études d'exportation d'un document LimSee3 vers SMIL, XHTML et aussi l'exportation d'un document avec un template.

8.2.1 Étude de l'exportation vers SMIL

Pour mieux comprendre les mécanismes d'exportation vers SMIL, nous allons utiliser l'exemple du document LimSee3 présenté dans la section 6.4 et le convertir dans le format de document SMIL. La Figure 9 montre la structure de l'exemple qui est composé de trois parties séparées *children*, *timing*, et *layout*.

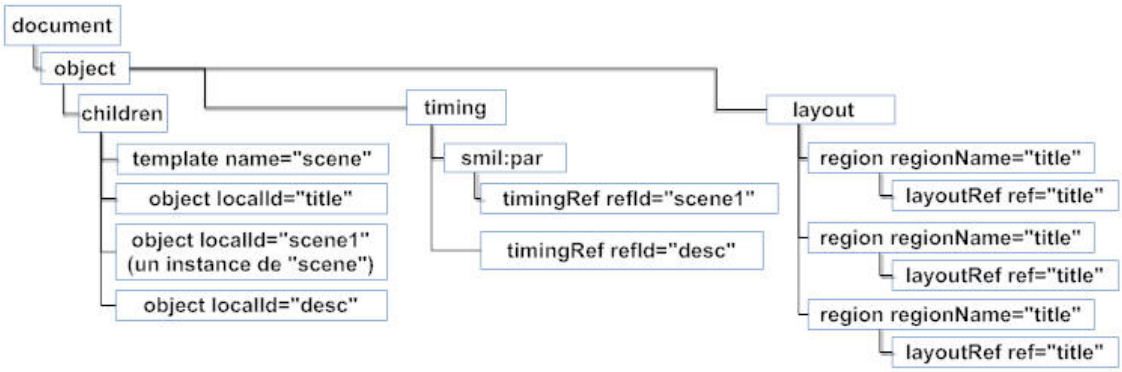

Figure 9 : La structure de l'exemple (Liste 1)

Exportation de la structure spatiale

SMIL définit la disposition dans la section *head* du document. LimSee3 ne définit pas explicitement le *root-layout*. En fait, c'est la section spatiale de l'objet de plus haut niveau qui joue le rôle de *root-layout*. Les attributs de *root-layout* correspondent aux attributs spatiaux de l'élément de plus haut niveau de la structure logique. La région de SMIL correspond exactement à la définition LimSee3 *smil:region*. La source SMIL de la section *layout* correspondante à l'exportation de l'exemple est listée ci-dessous (liste 2):

Liste 2: la section spatiale de SMIL

```
1. <layout type="text/smil-basic-layout">
2.   <root-layout id="rootLayout" width="500" height="300"/>
3.   <region left="auto" top="auto" width="100%" height="50"
   id="r_title"/>
4.   <region width="350" top="50" left="auto" height="250"
   id="r_image1"/>
5.   <region backgroundColor="green" showBackground="always"
        top="50" width="150" left="350" height="250" id="r_desc"/>
6. </layout>
```

Les objets médias ne sont pas associés aux définitions de région. Par contre, l'attribut *region* d'un objet indique la région dans laquelle il sera placé. En plus, l'élément *layoutRef* définit la référence d'une région vers un objet média.

Exportation de la structure temporelle

Le modèle de document LimSee3 s'appuie sur plusieurs éléments et attributs de SMIL. Les éléments *smil:par*, *smil:seq*, *smil:excl* sont identiques à ceux de SMIL. En plus, *smil:par* est le container de temps défaut pour l'élément *timing*. Dans SMIL, au contraire,

le container de temps défaut pour le *body* est la séquence (*seq*). C'est pourquoi nous intercalons un container *par* représentant la section timing entre *body* et son contenu. La liste 3 présente la source du corps de document exporté de l'exemple 6.4.

Liste 3: La section temporelle de SMIL

```
7.  <body id="body">
8.    <par>
9.      <text src="medias/title.txt" dur="indefinite" region="r_title"/>
10.       <par begin="3s" dur="8s">
11.         <img src="medias/scene1.jpeg" region="r_image1"/>
12.         <text src="medias/desc.txt" region="r_desc"/>
13.       </par>
14.     </par>
15.   </body>
```

Notons aussi que le *template* qui est un « document à trous » ne contient pas explicitement d'un élément multimédia ni d'une disposition (temporelle ou spatiale). Donc, nous l'ignorons dans l'exportation.

Pour l'instant, LimSee3 ne supporte pas les animations et les transitions. Si ces notions viennent enrichir LimSee3, cela risque de rendre l'exportation plus problématique.

Finalement, la Figure 10 résume le principe de l'exportation de LimSee3 vers SMIL.

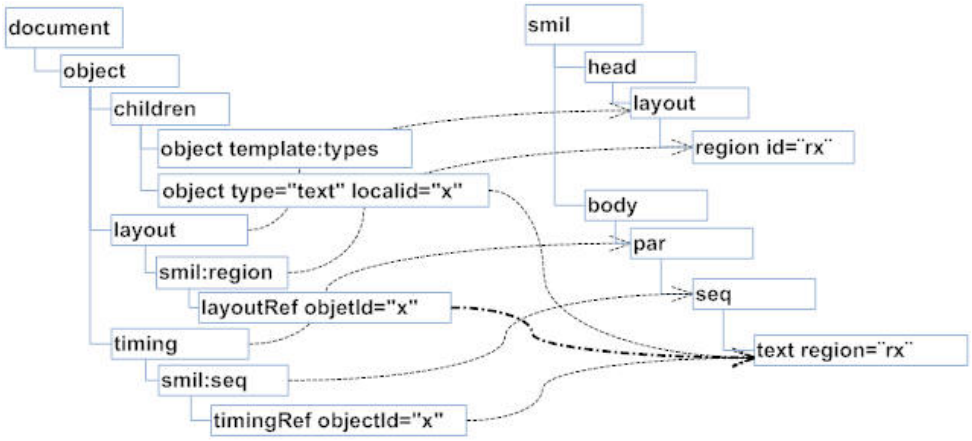

Figure 10: La correspondance entre le document LimSee3 et le document SMIL

Exportation vers XHTML+SMIL

L'autre format de sortie issu des travaux de SMIL est celui proposé par Microsoft, XHTML+SMIL [Schmitz 2000]. L'exportation de LimSee3 vers ce format est un peu

différente de précédent. Cependant, XHTML+SMIL utilise le mécanisme de positionnement spatial absolu (*position:absolute*) de CSS et l'inclut dans la structure temporelle de SMIL. Cette conversion fait perdre quelques attributs, car CSS n'est pas complètement compatible avec l'équivalent SMIL. Ainsi, dans cet exemple, l'attribut *showBackground* n'est pas généré parce que ce langage ne contient rien d'équivalent.

Liste 4: L'exportation de l'exemple vers XHTML+SMIL code

```
1. <html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:smil="http://www.w3.org/2001/SMIL20">
2.    <head>
3.    </head>
4.    <body>
5.       <smil:par>
6.          <smil:text src="medias/title.txt" dur="indefinite"
      style="position:absolute;left:auto;top:auto;width:500;height:50;" />
7.          <smil:par begin="3s" dur="8s">
8.             <smil:img  src="medias/img1.jpg"
      style="position:absolute;left:auto;top:50;width:350;height:250;" />
9.             <smil:text  src="medias/desc.txt"
      style="position:absolute;left:350;top:50;width:150;height:250;" />
10.         </smil:par>
11.      </smil:par>
12.   </body>
13.   </html>
```

Il est possible d'intégrer du code JavaScript qui récupère les événements générés par le navigateur et permet de contrôler la synchronisation des objets média.

8.2.2 Étude de l'exportation vers XHTML

Comme l'exemple ci-dessus l'a montré, la conversion d'un document LimSee3 vers SMIL ne pose pas de grande difficulté du fait de son emprunt des concepts de SMIL. Nous décrivons ici les problèmes d'exportation de LimSee3 vers le format XHTML plus CSS et JavaScript pour lequel les difficultés sont plus importantes puisque les synchronisations n'y sont pas explicitement définies et qu'elle sera découvert de recourir à un *ordonnanceur* (voir la section 9.3) couplé avec le navigateur. Il faut analyser les structures et les sémantiques de XHTML pour identifier les correspondances possibles entre les éléments deux langages.

La structure d'un document LimSee3 est très différente de celle d'un document XHTML.

Nous analysons ici les correspondances possibles entre des éléments LimSee3 et des éléments XHTML ainsi que les capacités d'intégration de JavaScript et CSS :

1. La structures logique : Les éléments objets complexes qui structurent des documents et groupent des sous-objets et des éléments médias correspondent aux étiquettes *div* et *span* de XHTML à décorer par des attributs CSS pour ce qui est des aspects spatiaux.
2. Les objets médias : De façon générale, l'élément textuel correspond au paragraphe (étiquette *p*) de XHTML. Les autres objets médias sont convertis vers XHTML par l'étiquette *img* (image), *object* ou *embed* (vidéo, audio, animation). XHTML ne supporte pas de descriptions de temps (les attributs *dur, begin, end,…*) ou des synchronisations entre des objets. De plus, un lecteur de document multimédia doit être capable de suivre et traiter les états de chaque élément média lors de l'exécution du document *(*e.g. *visible, invisible, begin, stop, outofsync,…).*

   Pour compenser cette lacune de XHTML+CSS, il faut utiliser un langage externe exécutable par les lecteurs Web. C'est le cas de JavaScript qui permet de récupérer et traiter les états des objets. Dans ce cas, le navigateur appelle les lecteurs de média externes (audio, vidéo, animation) qui agissent comme des plugins pour la présentation d'objets médias.
3. La disposition spatiale : Les éléments *layout*, *smil:region* basé sur SMIL sont compatibles avec le mécanisme de disposition absolue de CSS [Bulterman 2004, 281-283].
4. Les événements : Les événements sont divisés en deux catégories : les interactions d'utilisateur (les événements interactifs) comme *mouseclick, mouseover,…* et les événements non-interactifs comme *beginevent*, *endevent*, *outofsync,…* qui sont générés par le système pendant l'exécution. JavaScript est capable de récupérer la plupart des événements interactifs. Au contraire, JavaScript ne peut pas traiter certains événements générés au cours de l'exécution comme *outofsync, syncrestored* du fait que les navigateurs délèguent la visualisation des médias aux plugins qui ne génèrent pas ces événements.

   Pour pallier cette limité dans le cas des événements donc on peut calculer l'arrivé statiquement. C'est le cas des événements *begin*, *end*, *dur* qui sont liés la durée de l'exécution de média (vidéo, audio, animation). Lors de l'exécution, l'ordonnanceur peut utiliser des valeurs calculées pour déterminer et générer les événements du document. Cette approche risque de perdre certaines synchronisations entre des objets

médias comme des synchronisations qui utilisent les événements irrécupérables (*outofsync*, *syncrestored*).

5. Le temps : XHTML ne supporte pas d'expression de temps dans un document. Aussi un ordonnancement est généré à partir des attributs temporels (*dur*, *begin*, *end*). JavaScript contrôle les durées de présentation de chaque objet média, et génère des événements non-interactifs et la synchronisation entre des objets médias. En effet, les containers du temps (*par*, *seq*, *excl*) sont convertis à l'ordonnancement. Donc JavaScript peut les traiter.
6. Les liens : LimSee3 propose des liens traditionnels comme XHTML. En plus, il est basé sur SMIL en ajoutant le temps dans un lien qui n'est pas supporté dans XHTML. Nous utilisons JavaScript pour contrôler l'exposition des liens temporels.
7. Les transitions et les animations : XHTML ne supporte pas des transitions et des animations. Pour l'instant, LimSee3 ne les supporte pas. Mais cette extension envisagée pour LimSee3 risque de poser des problèmes lors de son exportation du fait de JavaScript peut contrôler des coordonnées d'un média.

L'analyse ci-dessus monte que la plupart d'éléments LimSee3 sont convertibles vers XHTML en ajoutant du code JavaScript et CSS ainsi que la possibilité de jouer des éléments multimédias sur la plupart de navigateurs avec des plugins.

8.2.3 Étude de l'exportation avec un template

Le template est un squelette de document qui contraint la structure globale du document. L'exportation peut tirer parti des templates en proposant un mécanisme de prétraitement adapté. Dans cette section, le *slideshow* est utilisé comme un exemple de document multimédia qui s'appuie sur un template.

Figure 11: La structure de slideshow

Le *slideshow* est un cas particulier de document multimédia dont la structure globale prédéfinie (Figure 11) comporte une suite ordonnée de transparents. Le contenu de chaque transparent est un document complet qui contient des structures logique, spatiale, et temporelle séparé et qui conforme au squelette de *slideshow*. Donc, l'exportation d'un

transparent réutilise la plupart de l'analyse du document général.

Pour le prétraitement d'un template, nous nous intéressons à la structure globale de template. À un instant donné, le *slideshow* dispose à l'écran un transparent et fournit diverses façons d'enchaîner la présentation vers un autre transparent (précédent, suivant, ou bien une page particulière). Dans le cas de XHTML, c'est à JavaScript de gérer les transparents et de récupérer les interactions d'utilisateurs pour générer les enchainements demandés. Dans le cas de SMIL, les transparents se situent dans une structure temporelle *« seq »*. Chaque transparent fournit un bouton « suivant » et un bouton « précédant » qui ont pour but de basculer vers le transparent suivant ou le transparent précédant. Cette structure définit le squelette de SMIL pour un transparent.

L'exportation avec un template permet de bien traiter les squelettes particuliers pour chaque application. En plus, elle permet aussi d'intégrer son résultat à l'application existant. C'est le cas de l'exportation de *slideshow* vers *Slidy* [Raggett 2005]. *Slidy* est une bibliothèque JavaScript qui génère et gère les transparents sous le format XHTML et les feuilles CSS spécifiques. L'exportation vers Slidy génère des sources XHTML en respectant les contraints de Slidy. Par exemple, Slidy définit un transparent par une étiquette *div* utilisé la classe CSS *slide :*

* … *

Le contenu de transparent est les éléments descendants de cette étiquette. L'ordonnanceur en JavaScript rajoute la disposition temporelle dans le document résultat.

En plus, cette approche profite des cadres d'application existants. Par contre, elle a la dépendance de cadre d'application externe. En plus, Il manque un mécanisme générique pour le problème d'exportation.

8.2.4 Bilan sur l'étude des besoins d'exportation

Les exemples ci-dessus ont illustré les deux catégories d'exportation nécessaires:

- Un document LimSee3 général défini avec des structures internes de LimSee3 pour générer les enchaînements demandés.
- Un document LimSee3 créé à partir d'un template LimSee3.

Dans le premier cas, le document peut être transformé directement vers un document

cible. Il faut chercher les correspondances entre les concepts élémentaires (image, vidéo, région,…) du document LimSee3 et du format cible. Puis, les relations entre la structure du document LimSee3 et le format cible doivent être spécifiées afin de déterminer le processus de conversion.

Dans le deuxième cas, le template donné par une structure globale permet de prétraiter l'exportation pour une application particulière. En plus, cela sépare bien le contenu et la structure de document.

Enfin, cette étude nous permet de dégager un ensemble de critères afin d'évaluer les différentes solutions pour le problème d'exportation :

- Adaptation à plusieurs formats : comme on l'a vu, les formats du document cible sont variés. Donc la solution doit être la plus adaptée à cette diversité de formats afin d'expérimenter un cadre d'application qui permet de créer facilement un nouveau transformateur pour un nouveau format.
- Extensibilité : l'exportation doit prendre en compte les changements et les évolutions de LimSee3 ou des formats cibles.
- Optimisation de résultat: l'exportation doit être capable d'optimiser le résultat obtenu pour chaque format cible et également pour chaque type de lecteur.
- Performance d'exportation: l'algorithme d'exportation doit être efficace.
- Facilité de conception : la proposition doit être facile à mettre en œuvre et également à créer un nouvel exportateur.

8.3 Approches possibles pour l'exportation

Dans cette section, nous décrivons deux approches que nous avons étudiées : la première s'appuie sur des convertisseurs Java qui traduisent la présentation interne de LimSee3 vers le format cible, la deuxième s'appuie sur un format intermédiaire puis le transforme vers le format cible avec une feuille de transformation XSLT.

8.3.1 Convertisseur Java

Cette approche profite de structures internes temporelles et spatiales qui sont disponibles dans l'implémentation de LimSee3 sous la forme d'une instance DOM contenant les structures brutes qui est maintenue pour la manipulation interne de l'éditeur. Les représentations temporelles et spatiales sont converties vers le document cible à l'aide de

classes Java. L'utilisation d'un langage de programmation comme Java facilite la programmation des calculs complexes de conversion comme la résolution de coordonnées absolues de région. Par exemple, le calcul d'ordonnancement, la manipulation sur l'arbre de la structure LimSee3. Enfin, cette solution réutilise emploie l'avantage de résultats de LimSee3.

Un convertisseur abstrait qui fait l'encapsulation de fonctions communes ainsi que des informations essentielles pour la conversion peut être conçu en utilisant l'API de LimSee3. Ce cadre d'application permet d'optimiser et de représenter les structures d'objets médias, spatiales et temporelles pour faciliter le processus de transformation. Il pourra être un squelette réutilisable pour chaque convertisseur spécifique.

La Figure 12 montre la structure de ce convertisseur Java qui contient : un convertisseur abstrait qui réalise des fonctions communes (voir la section 9.1) pour l'exportation ; des convertisseurs concrets qui sont spécifiques à chaque format cible. L'implémentation de LimSee3 fournit la structure complète du document et deux arbres spécifiques qui représentent la dimension spatiale et la dimension temporelle. Une table de hachage conserve la relation entre les objets, les définitions spatiales et les définitions temporelles. Le convertisseur utilise cette API pour manipuler le document en entrée.

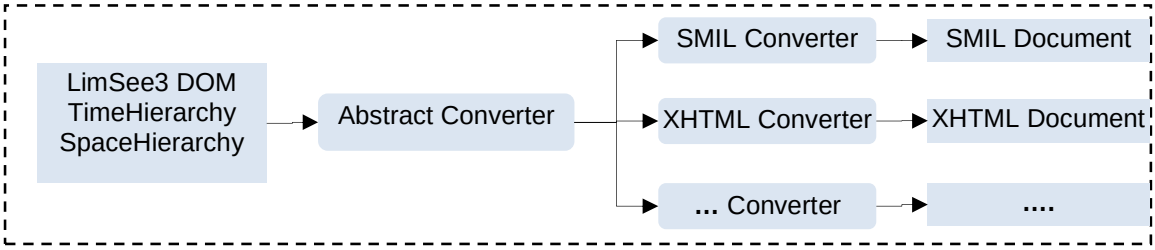

Figure 12: L'exportation avec la structure interne de LimSee3

L'avantage de cette approche est la réutilisation du cadre d'application comportant des fonctions communes. En plus, les calculs par Java rendent l'optimisation plus facile. Le cadre d'application est bien adapté aux formats textuels ou binaires. Cependant, un changement de format LimSee3 impose des changements sur chaque convertisseur. Donc cette solution est difficilement extensible lors de l'évolution de LimSee3.

8.3.2 Transformateur avec XSLT

Cette approche nécessite un format intermédiaire qui est utilisé pour faciliter la transformation vers chaque format cible (Figure 13). Le document intermédiaire est créé

par une feuille de style XSL ou bien basé sur le DOM LimSee3 et les structures internes temporelle et spatiale. Un processus d'optimisation est aussi appliqué avant de créer le document intermédiaire.

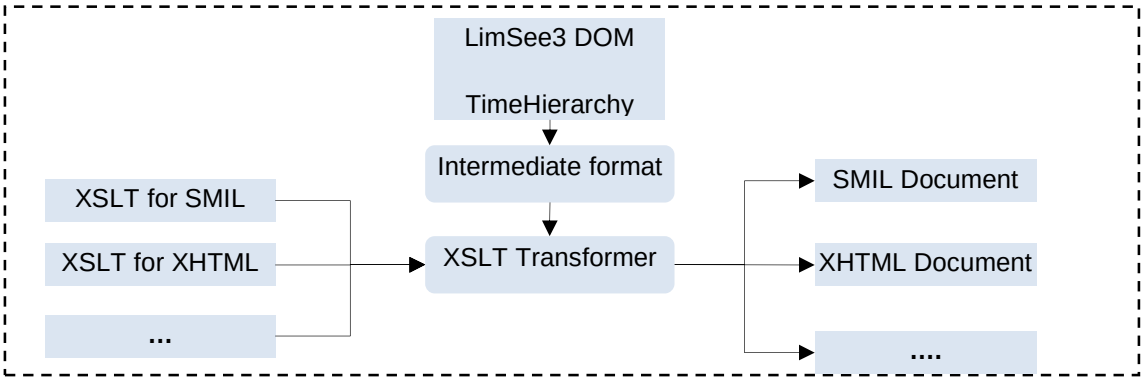

Figure 13: L'exportation par un format intermédiaire

À titre d'exemple, SMIL définit la disposition spatiale dans une section séparée, la structure principale du document est la structure temporelle, dont les feuilles sont les objets médias avec les références vers leur région. Par contre, XHTML s'appuie directement sur la structure logique du contenu et le document est décoré par des définitions CSS séparées. Pour permettre l'exportation vers des formats aussi différents, nous faisons le choix d'avoir le format intermédiaire qui est une forme réduite du format de document LimSee3 dans laquelle ont été effectués des prétraitements des relations complexes, des ordonnancements. La transformation vers les documents cible est ensuite exécutée par XSLT. Le choix de la structure de format de document intermédiaire doit prendre en compte la diversité des structures des formats de sortie et les possibilités de transformation offerte par XSLT.

L'API de LimSee3 est utilisée pour la manipulation du document en entrée du processus de génération du document intermédiaire.

Le format intermédiaire est décrit par la DTD donnée en annexe 2. Le principe du format intermédiaire est la représentation d'un document LimSee3 sous une autre forme qui facilite la manipulation par XSLT et prétraite des relations complexes de document LimSee3 comme la relation entre *regPoint*, *regAlign* et une région.

Le format intermédiaire se compose de : le *head* comporte les métadonnées ; la section *layout* qui présente des contenus spatiaux et leurs attributs ; la section *timing* qui présente des structures temporelles et des références vers des objets médias ; la section *media* qui contient des définitions d'objets médias et leurs attributs ; la section *references* qui

contient des références entres des objets médias et des dispositions spatiales. La Figure 14 présente graphiquement les différentes parties de ce format intermédiaire et la correspondance entre le document de LimSee3 et le format intermédiaire.

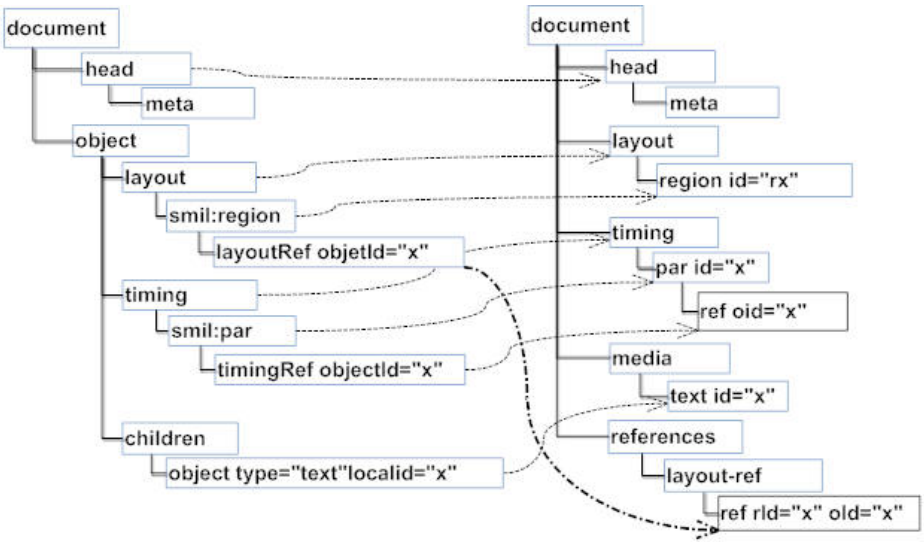

Figure 14: La correspondance entre le document LimSee3 et le format intermédiaire

L'avantage du format intermédiaire est que l'exportation est plus transparente et plus facile à déboguer car le format basé sur XML est lisible. Cette approche sépare bien le format de source et le format cible textuel. En particulier, un changement dans le format de LimSee3 n'affecte pas la transformation du document intermédiaire en document cible. En plus, une collection de thèmes peut être facilement créée pour enrichir les styles du document cible. Cependant, XSLT étant un langage déclaratif, il n’est pas flexible pour des calculs complexes. Donc l’optimisation est coûteuse. Il ne supporte pas non plus l’exportation vers des formats binaires.

8.3.3 Bilan sur les approches d’exportation

Nous avons proposé deux approches possibles pour le problème d’exportation. Il n'existe pas une solution parfaite pour ce problème. Le format intermédiaire s’adapte bien aux formats cible textuels ou basés sur XML. Par contre, le convertisseur Java permet de réaliser plus facilement des calculs complexes.

Ces deux approches en visant à générer un document à partir d’un document source LimSee3 permettent d’implémenter un seul cadre d’application. Pour la deuxième approche, un processus supplémentaire inclut la création une feuille XSLT et la transformation de document intermédiaire vers des formats cibles en appliquant cette

feuille.

La section suivant montre l'expérimentation des deux approches pour bien évaluer ces solutions.

9. Implémentation

9.1 Architecture

Les deux approches visent à générer un document à partir d'un document LimSee3. Donc, leurs implémentations peuvent partager des fonctions communes. L'architecture du cadre d'exportation (Figure 15) inclut :

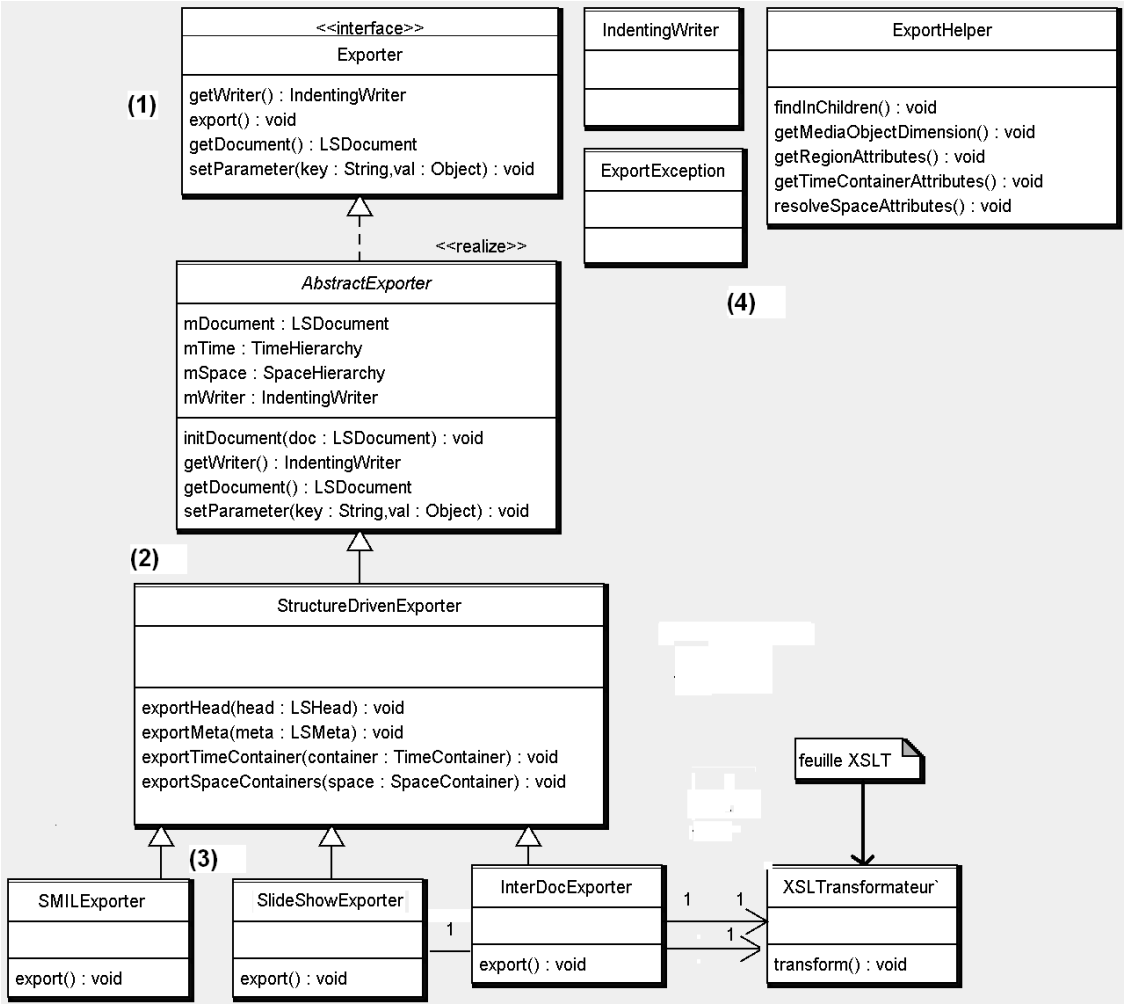

Figure 15: Le diagramme de classes d'exportation

(1). L'interface *Exporter* qui décrit des fonctions nécessaires pour l'exportation.
(2). Les classes abstraites *AbstractExporter* et *StructureDrivenExporter* qui encapsule les *fonctions communes* comme les fonctions de base pour le cadre d'exportation (*getWriter(), getDocument(), setParameter()*), les manipulations sur des structures temporelles, les fonctions pour résoudre des attributs spatiaux et temporels comme *regPoint*, *regAlign*, des expression de synchronisations (*endsync*, *begin*, *end*). En plus, *StructureDrivenExporter* fournit un squelette grâce aux parcours sur les structures temporel et spatial.
(3). Les classes concrètes qui implémentent le processus d'exportation pour chaque format cible. Les classes *SlideShowExporter* et *SMILExporter* sont les convertisseurs en Java pour l'exportation vers *slideshow et SMIL*. La classe *InterDocExporter* génère le document en format intermédiaire. La classe *XSLTransformater* a pour but d'appliquer la feuille XSL sur le document intermédiaire.
(4). Les classes utilitaires comme les classes *ExportHelper*, *IndentingWriter*.

Notons qu'un exportateur concret doit implémenter l'interface *Exporter*. Dont le cadre d'application peut le comprendre. En plus, Il peut hériter de l'*AbstractExporter* pour profiter des fonctions communes.

9.2 Exportation vers SMIL

LimSee3 s'appuie sur des concepts de SMIL : tous ses éléments et ses attributs sont compatibles et convertibles vers ceux de SMIL. L'étude 8.2.1 a présenté les correspondances entre des éléments LimSee3 et les celles de SMIL.

En plus, les structures internes de LimSee3 incluant *SpaceHierarchy* et *TimeHierarchy* qui sont le résultat de processus complexes pour résoudre les templates, les références et les instances peuvent être représentées par des structures logique, spatiale, et temporelle de SMIL. La classe *StructureDrivenExporter* facilite la transformation de structure LimSee3 vers une autre structure par un mécanisme abstrait. La classe *SMILExporter* hérite la classe *StructureDrivenExporter* est dédiée à l'exportation vers SMIL. Elle fait des conversions basées sur les correspondances entre les éléments LimSee3 et ceux de SMIL.

Quelques lecteurs de SMIL ne supportent pas le texte avec des styles ou le présentent incorrectement. Un processus supplémentaire de conversion de texte avec des styles vers

l'image est ajouté dans l'exportation. C'est une option pour l'utilisateur.

L'exportation avec une feuille XSL est équivalente sauf la conversion de texte avec des styles vers l'image.

9.3 Exportation vers XHTML+JavaScript+CSS

L'étude 8.2.2a a analysé la correspondance entre les éléments LimSee3 et ceux du triple langage XHTML+JavaScript+CSS. Nous implémentons : (1) un exportateur vers XHTML soit par un exportateur en Java, soit par un exportateur vers un document intermédiaire en plus d'une feuille XSLT ; des feuilles de style CSS qui permettent d'enrichir les apparences de document cible; (2) l'ordonnancement et (3) un ordonnanceur JavaScript qui permet de synchroniser les objets médias et de contrôler le document, et les interactions d'utilisateurs.

9.3.1 XHTML+CSS

L'exportateur génère le code XHTML statique qui comprend : des structures logique et spatiale qui se composent de définitions de disposition et de décoration de document multimédia sous la forme XHTML et CSS. Cette partie est générée par code Java ou est le résultat de transformation d'un document intermédiaire.

L'exportateur utilise les règles analysées ci-dessus (la section 8.2.2) pour générer les balises XHTML+CSS.

9.3.2 Ordonnancement

L'ordonnancement généré par l'exportateur et utilisé par l'ordonnanceur JavaScript est un fragment XML qui contient un tableau de temps et des dépendances entre des éléments XHTML, des containers du temps (*seq, par, excl*). Il permet de représenter la structure temporelle d'un document LimSee3 par un graphe qui contient des nœuds et des relations entre nœuds. Il est conforme au schéma DTD donné dans l'annexe 1.

- Il y a deux types de *nœud* : le *nœud XHTML* qui associé à un élément XHTML/CSS, des attributs temporels, des comportements d'élément, et des états pendant l'exécution. Par exemple :

  <object id='desc' isHTML='1' begin='3' dur='1000' fill='remove'>

Cet exemple représente un élément XHTML/CSS ‘desc’ qui s’active à ‘3ms’ pendant ‘1000ms’ relativement à son conteneur. En plus, il va être caché après sa désactivation.

Le *nœud virtuel* qui représente un container temporel (*seq, par, excl*). Il vise la facilité de synchronisation et propagation des événements entre les éléments. Par exemple,

<object id="par_1" hasHTML='0' dur="2s" ensync="x1" >

Ce nœud représente un container parallèle qui active à ‘0ms’ pendant ‘2s’ relativement à son container. Il désactive et génère l’événement *end* quand l’élément x1 désactive.

- La *relation* définit la relation entre un nœud source et un nœud cible. Deux nœuds sont en relation si un événement du nœud source affecte le nœud cible. Par exemple, l’événement ‘*end*’ (après son activation de 1000ms) du nœud ‘*finish*’ impose la fin (‘*end*’) au nœud ‘*desc*’ et le démarrage au nœud ‘*cloture*’. Cette relation se divise en deux triggers : *end (A)* → *end (B)* et *end (A)* → *begin (C)*. Elle est représentée par le fragment d’ordonnancement suivant :

```
<object id=‘finish’ dur='1000' fill='remove' >
        <end_listener>
                <item id=‘desc’ target='end'/>
                <item id=‘cloture’ target='begin'/>
        </end_listener>
</object>
```

Le générateur d’ordonnancement permet de gérer événements et les synchronisations définies par LimSee3 incluant les événements *non-interactifs* (begin, end, beginEvent, endEvent, repeatEvent, repeat, mediacomplete, outofsync, syncrestored, pause, resume, reverese, seek, reset, timeerror) et *interactifs* (activateEvent, click, mouseover, mouseout, mousedown, mouseup, mousemove, focusInEvent, focusOutEvent, outOfBoundsEvent, inBoundsEvent) [Bulterman 2004, p 285-326].

En général, la synchronisation utilise la syntaxe X=I.Y+n où I est l’élément générant l’événement source Y qui est *non-intératif* ou *intératif*, n est un nombre représentant le temps, et X est l’événement cible de type *intératif* ou *non-intératif*. La Figure 16 est un exemple de synchronisation entre deux objets. Sur le nœud I2,

supposant que la synchronisation se forme *begin=I1.click + 5s*, on a une relation *click (I1)* → *begin (I2)* et l'événement *begin* effectif sur I2 se passera après 5s de l'événement click sur I1. Ce retard est contrôlé par JavaScript.

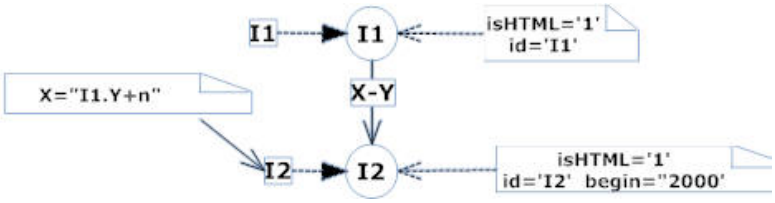

Figure 16: La représentation de l'ordonnancement de syncbase

Les éléments XHTML écoutent les événements interactifs et les transmettent aux nœuds correspondants. Ce sont les nœuds source. Les nœuds dépendants de cette interaction sont les nœuds cible. La relation est définie selon le type de dépendance. Par exemple, un clic sur un bouton X fait arrêter l'élément Y. La relation est dont *click-end*.

Pour bien gérer les nœuds, nous ajoutons deux types d'événement : *endofchild*, et *beginofchild.* Ils sont utilisés par le container parallèle et le container exclusif et décris dans les sous-sections suivantes.

La structure de LimSee3 est transformée par l'exportateur Java vers l'ordonnancement selon trois containeurs de temps différents.

- Containeur « *seq* » est transformé en un containeur-nœud (nœud virtuel) et une suite de nœuds XHTML qui ont la dépendance « *end-begin* ». Le container-nœud a la relation « *begin-begin* » avec le premier nœud XHTML. La figure 17a montre la conversion vers l'ordonnancement d'un container « *seq* » avec deux descendants.

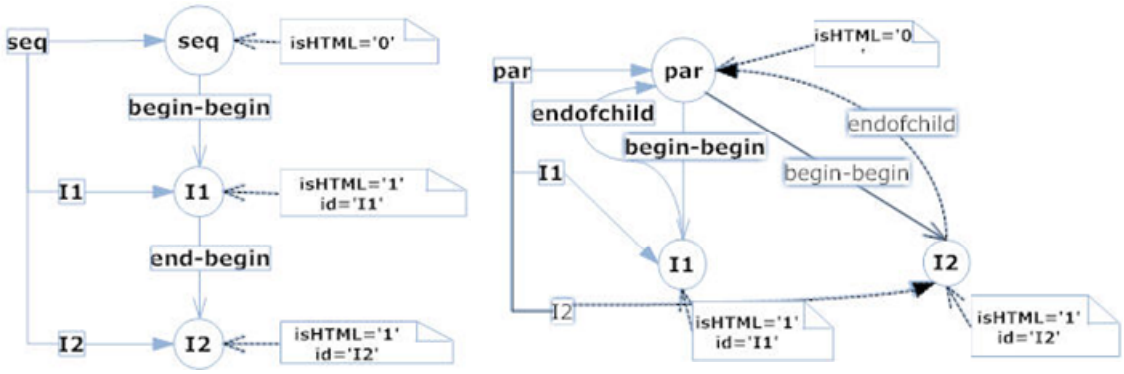

Figure 17: La représentation d'ordonnancement de (a) container "seq" et (b) container "par"

- Containeur « *par* » est transformé en un containeur-nœud (nœud virtuel) et des nœuds XHTML. L'activation de containeur-nœud impose des nœuds XHTML activés. Chaque nœud XHTML a une relation « *endofchild* » avec le containeur-nœud. Cette relation vise à signaler au containeur-nœud la terminaison de son descendant. Lors de réception de cet événement, le containeur-nœud le traite en dépendant le '*endsync*', et les définitions du temps (*dur*, *end*). La figure 17b montre la conversion vers l'ordonnancement d'un container « *par* » avec deux descendants.

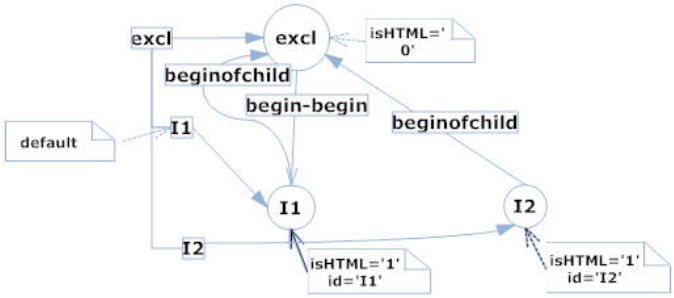

Figure 18: La représentation d'ordonnancement de container "excl"

- Containeur « *excl* » est transformé en un containeur-nœud (nœud virtuel) et des nœuds XHTML. Le containeur-nœud se compose d'une liste de relation « *excl* » qui contiennent tous les nœuds XHTML. Chaque nœud XHTML a une relation « *beginofchild* » avec le containeur-nœud. Cette relation vise à signaler le containeur-nœud le commencement de son descendant. Lors de réception de cet événement, le containeur-nœud force les autres descendants '*end*'. La Figure 18 montre la conversion vers l'ordonnancement d'un container « *excl* » avec deux descendants. Nous supposons que l'élément I1 est le défaut.

La section ci-dessous montre l'utilisation d'ordonnancement.

9.3.3 Ordonnanceur JavaScript

L'ordonnanceur en JavaScript utilise l'ordonnancement pour contrôler des éléments XHTML et faire la synchronisation entre des objets médias. Il récupère aussi des interactions de l'utilisateur et les transmet vers les éléments qui en dépendent et détecte des événements générés par le système si c'est possible.

À partir de l'ordonnancement généré, l'ordonnanceur construit un graphe (Figure 19) qui représente les nœuds et les relations entre les éléments XHTML. Ce graphe est construit et démarré dès que le navigateur charge le document. L'élément *objects* correspond à un

graphe. Chaque élément *object* d'ordonnancement est lié un nœud. Les éléments *item* dans chaque élément *xxx_listener* correspondent à une relation.

L'ordonnanceur contrôle des activations et des désactivations d'éléments XHTML/CSS selon l'ordonnancement en laissant le navigateur faire la disposition spatiale. Il récupère les interactions humaines et des événements de système et les envoie vers des nœuds. L'élément racine (*root_id*) est démarré pour activer le graphe dès que le document est chargé.

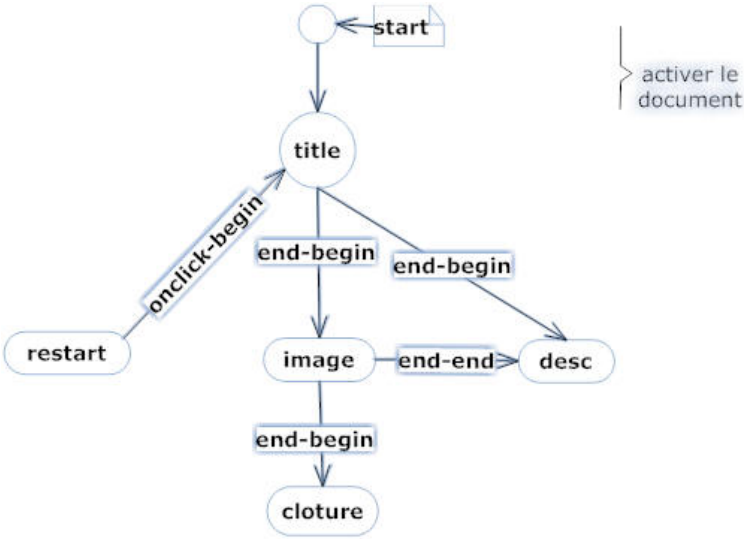

Figure 19: Un exemple de modèle de graphe de ordonnanceur JavaScript

Chaque nœud peut générer des événements et les propager vers les nœuds d'écoutant. Dès la réception d'un événement, le nœud d'écoute la traite en fonctionnant l'action cible. En plus, dans chaque élément XHTML est installé un « *écouteur* » qui récupère des événements interactifs. Le nœud correspondant à cet élément traite ces événements et les transfère vers ceux qui en dépendent.

9.3.4 Bilan sur l'exportation vers XHTML

Cette exportation permet d'exporter un document LimSee3 vers un document XHTML, CSS avec un ordonnancement. L'ordonnancement est défini donc un format indépendant de celui du document LimSee3. Nous avons créé un ordonnanceur qui constitue une bibliothèque de JavaScript exécuté dans le navigateur et interprète l'ordonnancement en gérant le code XHTML/CSS.

Le tableau ci-dessous compare ces deux approches enfin de les évaluer selon les critères définis dans la section 8.2.4 :

Critère	Transformateur avec XSLT	Convertisseur Java
Adaptation à plusieurs formats	+	++
Optimisation du résultat	+	++
Extensibilité de Limsee3	++	+
Performance d'exportation	+	+
Facilité de conception	++	+

Le convertisseur Java est adapté aux formats de présentation multiples et permet d'optimiser le résultat du fait que le langage de programmation Java est plus flexible que XSL. Par contre, le format intermédiaire dans la transformation avec XSLT est indépendant des évolutions de LimSee3 et des formats cible. Donc, il est plus adapté à l'extensibilité de LimSee3 et des changements de format cible dans le futur. Enfin, pour créer un nouvel exportateur, la programmation avec XSL est plus facile que celle en Java car cette dernière demande une bonne connaissance d'API et des structures internes de LimSee3.

9.4 Exportation avec un template

Comme on l'a vu dans la section 8.2.3, l'exportation avec un template prétraite le squelette de document. Nous avons effectué l'exportation de *slideshow* de LimSee3 vers *Slidy*. Une autre expérimentation intègre l'ordonnanceur dans Slidy avec un générateur d'ordonnancement pour *slideshow*.

L'exportation de *slideshow* est divisée à deux étapes : (1) l'exportation de squelette où la structure globale est convertie vers le format de Slidy. Cette étape vise à intégrer l'ordonnanceur de l'étape précédente à Slidy. En effet, l'ordonnancement est divisé à plusieurs sections qui correspondent à chaque transparent. Lors de l'activation d'un transparent, l'ordonnanceur prend l'ordonnancement correspondant et l'active. L'ordonnanceur s'est également arrêté dès la désactivation d'un transparent ; (2) l'exportation de contenu d'un transparent. Cette étape réutilise complètement les fonctions d'exportation de XHTML dans la section précédente (le code XHTML/CSS, l'ordonnancement).

Dans cette réalisation, nous avons implémenté l'exportateur en Java pour générer le squelette de *slideshow* et réutilisons la partie d'exportation d'ordonnancement et de document intermédiaire vers le format XHTML+CSS+JavaScript, et la feuille XSL pour transformer le document intermédiaire vers le code XHTML/CSS. La bibliothèque

supplémentaire en JavaScript est créée pour l'intégration de l'ordonnanceur à Slidy.

10. Résultats

L'objectif de la mise en place de service d'exportation de LimSee3 a été étudié et expérimenté pour différents formats de présentation : SMIL, XHTML+CSS+JavaScript et l'application *slideshow*. Nous avons proposé deux approches d'exportation : le convertisseur Java et le transformateur avec XSLT.

Sur la partie d'implémentation, nous avons conçu un cadre global d'application qui sert au service d'exportation et qui permet de faciliter la création de nouveaux exportateurs basés sur une ou l'autre des deux approches. En fait, ce cadre fournit un mécanisme abstrait et des fonctions communes qu'un nouvel exportateur peut réutiliser.

En plus, nous avons défini le format intermédiaire représentant le document LimSee3 sous une forme réduite et des feuilles XSL organisé par une bibliothèque réutilisable qui permet de manipuler facilement les documents intermédiaires.

Nous avons implémenté un ordonnanceur en JavaScript qui est capable d'exécuter un document XHTML en appliquant un ordonnancement défini donc un document XML conforme à une DTD d'ordonnancement que nous avons défini. Cet ordonnanceur, couplé avec le navigateur, implémenté sous la forme d'une bibliothèque de JavaScript, permet de rajouter les dispositions temporelles dans un document XHTML.

L'exportation de document LimSee3 vers Slidy montre la possibilité d'utilisation d'un template pour éditer et exporter des applications particulières. Le résultat intégré à Slidy permet de représenter les transparents avec les expressions de temps.

11. Conclusion

Dans ce mémoire, nous avons présenté les études liées à la technique d'édition des documents multimédias, des propositions de solution pour le service d'exportation de document LimSee3 vers les formats de présentation, et des expérimentations de ces solutions.

L'outil d'édition est un des trois problèmes concernant les documents multimédias à côté du format de représentation, et du lecteur du document. Le résultat de l'étape d'édition

doit fournir un document donc un format de présentation multimédia comme SMIL, Flash, ou MPEG4. Nous expérimentons les possibilités de représenter le document multimédia dans : (1) un format cible dédié au la présentation de document multimédia (SMIL) ; (2) un format qui ne supporte pas explicitement la disposition temporelle (XHTML + CSS + JavaScript) ; (3) une application spécifique qui utilise le template (*slideshow*). Ces expérimentations nous permettent de parcourir le problème d'exportation.

Le format intermédiaire conformité de DTD permet d'exprimer le document LimSee3 sous une forme réduite. Il réduit la complexité de concevoir un nouvel exportateur par la création d'une feuille XSL, et s'adapte également aux évolutions de LimSee3 dans le futur.

Nous proposons aussi la solution pour l'exécution de document multimédia au-dessus de navigateur Web. C'est le cas d'exportation vers XHTML+CSS+JavaScript avec l'ordonnancement conformité de DTD. L'ordonnanceur couplé avec le navigateur interprète cet ordonnancement. Donc l'ordonnancement et son interpréteur peut rajouter la disposition temporelle dans un document XHTML. Il est totalement indépendant de LimSee3.

Le template est une approche orientée application en facilitant la création d'un document multimédia dans un domaine particulier. La limitation de cette méthode est le manque d'un mécanisme générique de traiter les templates pour toutes les applications ou pour certaines applications.

La transformation est mise en place le plus possible pour SMIL car LimSee3 l'emprunte les concepts. Ce n'est pas totalement possible dans le cas de l'exportation vers XHTML par suite du manque d'expression de temps de XHTML et des incompatibilités entre leurs mécanismes de disposition. Pour l'exportation vers autre modèle de document, il faut déterminer la correspondance sémantique entre le modèle de document LimSee3 et le modèle cible. Donc, on peut réutiliser des fonctions proposées par le cadre d'application.

12. Perspective

Ce mémoire présente une expérimentation pour le service d'exportation. Elle fournit les bases à partir desquelles il est possible de créer les documents multimédias assez complexes. En plus de ces résultats, plusieurs perspectives à ces travaux sont

envisageables :

1. Template : Le template vise à définir une structure globale de document multimédia pour les applications particulières. Notre approche traite individuellement les templates qui dépendent des applications cible. Pour aller plus loin, il faut définir un triplet du template, l'interpréteur de template pour l'éditeur, et l'interpréteur de template pour l'exportateur. Il nous manque un mécanisme de traitement des templates génériques.
2. Adaptation à l'environnement de déploiement : de nos jours, les applications multimédias sont largement utilisés sur différents terminaux (les ordinateurs, les projecteurs les portables…), avec les différents débits (la connexion par câble optique, par réseau GPRS…). L'exportation doit permet aux utilisateurs de choisir des configurations cible. Il faut donc analyser et extraire des contenus liés aux paramètres entrés. Le résultat sera optimal dans le système cible.
3. L'exportation vers Flash, MPEG4 : les formats propriétaires posent des problèmes de copyright. Par contre, les lecteurs de Flash et MPEG4 sont disponibles sur la plupart des ordinateurs. Il faut étudier bien le modèle de document Flash ou MPEG4 donc on peut déterminer les correspondances entre le modèle de document LimSee3 et celui de Flash ou MPEG4. Puis, il faut créer un exportateur pour chaque langage en héritant le cadre d'exportation.
4. Le format de document LimSee3 s'appuie très fortement les concepts de SMIL. Il est peu incompatible avec autres formats (e.g. CSS) ce qui rend complexe et incomplet sa transformation vers ces formats. Il faut améliorer le format de LimSee3 pour qu'on résolve ces conflits

Bibliographie

Article

[Bailey 1998] B. Bailey, J. A. Konstan, R. Cooley, and M. Dejong . *Nsync - A Toolkitfor Building interactive Multimedia Presentations*. ACM international conference on Multimedia. 1998.

[Boll 2000] S. Boll, W. Klas, U. Westermann. *Multimedia Document Models*. University of Ulm, Germany, 2000.

[Buchanan 2005] M.C. Buchanan, P.T. Zellweger. *Automatic Temporal Layout Mechanisms Revisited.* ACM Transactions on Multimedia Computing, Communications, and Applications. 2005.

[Bulterman 2004] D.C.A Bulterman. *SMIL 2.0, Interactive Multimedia for Web and Mobile Devices.* Springer, 2004.

[Bulterman 2005] D.C.A Bulterman and L. Hardman. *Structured Multimedia Authoring*. ACM Transactions on Multimedia Computing, Communications, and Applications. 2005.

[Deemter 2000] K.V. Deemter, R. Power. *Authoring Multimedia Documents using WYSIWYM Editing*. Association for Computational Linguistics, 2000.

[Deltour 2005] R.Deltour, N.Layaïda, D.Weck. *LimSee2: A Cross-Platform SMIL Authoring Tool*. ERCIM News, July 2005.

[Deltour 2006a] R. Deltour, C. Roisin. *The LimSee3 Multimedia Authoring Model*. ACM Symposium on Document Engineering. 2006.

[Deltour 2006b] R. Deltour, A. Guerraz, C. Roisin. *Multimedia Authoring for CoPs*. TEL-CoPs'06, 2006.

[Jourdan 2001] M. Jourdan, C. Roisin, L. Tardif. *Constraint Technique for Authoring Multimedia*. Kluwer Academic Publishers, 2001 .

[Jourdan 2004] M. Jourdan, N. Layaïda, C. Roisin. *Le temps dans les documents.* Techniques de l'Ingénieur, 2004.

[Lee 2000] D. Lee, W.W. Chu. *Comparative Analysis of Six XML Schema Languages.* University of California, Los Angeles, 2000.

[Mikáč 2006] J. Mikáč, C. Roisin, R. Deltour. *LimSee3 Document Model v1.03.* http://limsee3.gforge.inria.fr/public-site/docs/LimSee3-document-model.html, 2006

[Murata 2000] M. Murata, D. Lee, M. Mani. *Taxonomy of XML Schema Languages using Formal Language Theory*. Extreme Markup Languages. 2000.

[Rabin 1996] M.D. Rabin, M.J. Burns. *Multimedia Authoring Tools*. AT&T Bell Laboratories, 1996.

[Rossum 1993] G. van Rossum, J. Jansen, K.S. Mullender and D.C.A. Bulterman. *CMIFed: A Presentation Environment for Portable Hypermedia Documents*. ACM Multimedia '93, Anaheim, Aug '93, 183 – 188

[Schmitz 2000] P.Schmitz. *HTML+SMIL Language Profile.* 2000.

[Schmitz 2001] P.Schmitz. *The SMIL 2.0 Timing and Synchronization model.* Microsoft, 2001.

[Soares 2004] L.F.G. Soares, G.L. De S. Filho, R.F. Rodrigues, and D.C. Muchaluat. *Versioning Support in the HyperProp System*. Multimedia Tools and Applications, Springer Netherlands, 2004

[Thuong 2003] T.T Thuong. *Modélisation et traitement du contenu des médias pour l'édition et la présentation de documents multimédias*. Thèse, 2003.

[Vlist 2001] E. van der Vlist. *Comparing XML Schema Languages.* O'Reilly Media, Inc., 2001.

Site Web

[Adobe] *Adobe* . http://www.adobe.com.

[Ambulant] *Ambulant*. http://www.cwi.nl/projects/Ambulant/distPlayer.html.

[Authorware] *Adobe Authorware*. http://www.adobe.com/products/authorware.

[Clark 2003] J. Clark, MURATA Makoto. *Relax NG*. http://www.relaxng.org/, 2003.

[Dicrector] *Adobe Director*. www.adobe.com/products/director.

[DSD2] DSD2. *Document Structure Description.* http://www.brics.dk/DSD/.

[Flash] *Adobe Flash.* http://www.adobe.com/products/flash.

[GRiNS] *GRiNS*. http://www.oratrix.com/GRiNS.

[LimSee2] LimSee2. http://wam.inrialpes.fr/software/limsee2.

[LimSee3] LimSee3. http://limsee3.gforge.inria.fr.

[Madeus] *Opéra, Madeus.* http://opera.inrialpes.fr/Madeus-Editor.html. 2002.

[Meyer 2006] E.A. Meyer. *Simple Standards-Based Slide Show System (S5).*

http://s5project.org. 2006.

[Mikáč 2006] J. Mikáč, *LimSee3*, http://limsee3.gforge.inria.fr/public-site/docs/Pub-utilisateur.htm. 2006.

[MPEG4] Moving Picture Experts Group. http://www.chiariglione.org/mpeg.

[Palette] Palette. http://palette.ercim.org.

[PowerPoint] *Microsoft PowerPoint*. http://office.microsoft.com/powerpoint.

[QuickTime] *QuickTime*. http://www.apple.com/quicktime.

[Raggett 2005] D. Raggett. *Slidy*. www.w3.org/Talks/Tools/Slidy. 2005.

[RealPlayer] *RealPlayer*. http://www.real.com.

[W3C.DTD] *W3C. DTD*. http://www.w3.org/XML/1998/06/xmlspec-report.htm.

[W3C.SMIL] *W3C. SMIL. Synchronized Multimedia*. http://www.w3.org/AudioVideo.

[W3C.SVG] *W3C. SVG*. http://www.w3.org/Graphics/SVG.

[W3C.XHTML] *W3C. XHTML*. http://www.w3.org/TR/xhtml1.

[W3C.XLink] *W3C. XLink*. http://www.w3.org/TR/xlink.

[W3C.XML] *W3C. XML*. http://www.w3.org/XML.

[W3C.XMLSchema] *W3C. XMLSchema*. http://www.w3.org/XML/Schema.

[W3C.XPath] *W3C. XPath*. http://www.w3.org/TR/xpath.

[W3C.XPointer] *W3C. XPointer*. http://www.w3.org/TR/xptr.

[W3C.XQuery] *W3C. XQuery*. http://www.w3.org/TR/xquery.

[W3C. XSLT] *W3C. XSLT*. http://www.w3.org/TR/xslt.

[WAM] *WAM*. http://wam.inrialpes.fr/software/limsee2. 2005.

Annexe

Annexe 1: Le DTD d'ordonnancement

```
<!-- ================================================================ -->
<!-- This DTD schema aims to define the JSPlayer's schedule syntax. -->
<!-- All the schedule should be conformed with this schema. -->
<!-- ================================================================ -->

<!--
Version 0.2: created.
Change log from version 0.2
============================
-->

<!-- =========Entities============================================ -->
<!ENTITY % lsplayer.id      "CDATA  #IMPLIED " >

<!-- This objects element is the root of the schedule. It uses to build
a graph in which represents the document. When the graph is activated,
the document is executing-->
<!-- === Objects ===================================================== -->
<!ELEMENT objects (root_id?, object+)>

<!-- RootID refers to an object within the schedule. It uses to
activate the graph. -->
<!-- === RootID ====================================================== -->
<!ELEMENT root_id EMPTY>
<!ATTLIST root_id
%lsplayer.id;
>

<!-- Object represents a node. It contains a list of chilren (for time
container like par, seq, excl), and 14 types of listener who refers to
other objects-->
<!-- === Object ====================================================== -->
<!ELEMENT object (child*, begin_listener?, end_listener?,
endofchild_listener?,
beginofchild_listener?, click_listener?, activateEvent_listener?,
mouseover_listener?, mouseout_listener?, mouseup_listener?,
mousemove_listener?, focusInEvent_listener?, forcusOutEvent_listener?,
outOfBoundsEvent_listener?, inBoundsEvent_listener?)>
<!ATTLIST object
%lsplayer.id;
<!--has an correspondent HTML element. Normally, time container is a
fade node and hasn't a correspondent HTML element.-->
hasHTML     "true|false" 'false'
dur         CDATA   #IMPLIED
repeatCount CDATA   #IMPLIED
repeatDur   CDATA   #IMPLIED
begin       CDATA   #IMPLIED
end         CDATA   #IMPLIED
repeat      CDATA   #IMPLIED
min         CDATA   '0'
max         CDATA   'indefinite'
```

```
restart     (always|whenNotActive|never|default)          'default'
restartDefault  (inherit|always|never|whenNotActive)
'inherit'
fill        (remove|freeze|hold|transition|auto|inherit)   'inherit'
>
<!-- Defines a child in which its identity associates with a object
in the graph. -->
<!-- === Child ================================================== -->
<!ELEMENT child EMPTY>
<!ATTLIST child
%lsplayer.id;
>

<!--Each listener contains a list of items-->
<!-- === Listener ================================================ -->
<!--the begin event on source object effects to the listener-->
<!ELEMENT begin_listener (item+)>

<!--the end event on source object effects to the listener-->
<!ELEMENT end_listener (item+)>

<!--the endofchild event on source object effects to the listener-->
<!ELEMENT endofchild_listener (item)>

<!--the beginofchild event on source object effects to the listener-->
<!ELEMENT beginofchild_listener (item)>

<!--the click event on source object effects to the listener-->
<!ELEMENT click_listener (item+)>

<!--the activate event on source object effects to the listener-->
<!ELEMENT activateEvent_listener (item+)>

<!--the mouseover event on source object effects to the listener-->
<!ELEMENT mouseover_listener (item+)>

<!--the mouseout event on source object effects to the listener-->
<!ELEMENT mouseout_listener (item+)>

<!--the mouseup event on source object effects to the listener-->
<!ELEMENT mouseup_listener (item+)>

<!--the mousemove event on source object effects to the listener-->
<!ELEMENT mousemove_listener (item+)>

<!--the focusin event on source object effects to the listener-->
<!ELEMENT focusInEvent_listener (item+)>

<!--the focusout event on source object effects to the listener-->
<!ELEMENT focusOutEvent_listener (item+)>

<!--the outofbounds event on source object effects to the listener-->
<!ELEMENT outOfBoundsEvent_listener (item+)>

<!--the inbounds event on source object effects to the listener-->
<!ELEMENT inBoundsEvent_listener (item+)>
```

```
<!-- === Item ========================================================== -->
<!--target present the action in the listener when it receives an
event-->

<!ELEMENT item ()>
<!ATTLIST item
%lsplayer.id;
target  "begin|end|pause|stop|endsync|non_specific" #IMPLIED
>
<!-- ================================================================== -->
```

Annexe 2: Le DTD de format intermédiaire

```
<!--
==================================================================== -->
<!--This is the DTD of LimSee3 intermediate format, version 0.3    -->
<!-- Conforming documents should declare the following doctype:    -->

<!-- <!DOCTYPE document SYSTEM
"http://ns.inria.fr/limsee3/intermediate.dtd"> -->
<!-- (NOTICE: the DTD will be actually uploaded to the server when it
reaches version 1.0)     -->

<!--
  Change log from version 0.2
  ============================
  * version 0.3:  - in the region, regPoint element, regPoint
attributes and regAlign attribute are added.
                  - Adds timing attributes for area element  -->
<!--
==================================================================== -->

<!ENTITY % namespace 'xmlns  CDATA  #FIXED
"http://ns.inria.fr/limsee3"'>
<!ENTITY % common.attr   "xml:lang  CDATA  #IMPLIED " >
<!ENTITY % lock          "lock  (locked|unlocked)  #IMPLIED"  >
<!ENTITY % system.attrs
       "systemBitrate               CDATA           #IMPLIED
        systemCaptions              (on|off)        #IMPLIED
        systemLanguage              CDATA           #IMPLIED
        systemOverdubOrSubtitle     (overdub|subtitle) #IMPLIED
        systemRequired              CDATA           #IMPLIED
        systemScreenSize            CDATA           #IMPLIED
        systemScreenDepth           CDATA           #IMPLIED
        systemAudioDesc             (on|off)        #IMPLIED
        systemOperatingSystem       NMTOKEN         #IMPLIED
        systemCPU                   NMTOKEN         #IMPLIED
        systemComponent             CDATA           #IMPLIED
        system-bitrate              CDATA           #IMPLIED
        system-captions             (on|off)        #IMPLIED
        system-language             CDATA           #IMPLIED
        system-overdub-or-caption   (overdub|caption) #IMPLIED
        system-required             CDATA           #IMPLIED
        system-screen-size          CDATA           #IMPLIED
        system-screen-depth         CDATA           #IMPLIED"
>
```

```
<!ENTITY % regAlign.attr "regAlign (topLeft | topMid | topRight |
midLeft | center | midRight | bottomLeft | bottomMid | bottomRight)
#IMPLIED " >

<!-- === The root element: document
========================================================== -->

<!ELEMENT document (head?, layout?, timing?, references?, media?) >

<!-- === Head elements: head & meta
========================================================== -->
<!ELEMENT head (meta)* >
<!ATTLIST head
  %common.attr;
  %lock;
>

<!ELEMENT meta EMPTY >
<!ATTLIST meta
  name    CDATA #REQUIRED
  content CDATA #REQUIRED
  %lock;
  %common.attr;
>

<!-- === layout elements: layout, region, & regPoint
========================================== -->

<!ELEMENT layout region >
<!ATTLIST layout
  type    CDATA  #FIXED  'text/smil-basic-layout'
>

<!ELEMENT region (region|regPoint)* >
<!ATTLIST region
  id                        ID     #REQUIRED
  bottom                    CDATA   'auto'
  left                      CDATA   'auto'
  right                     CDATA   'auto'
  top                       CDATA   'auto'
  height                    CDATA   'auto'
  width                     CDATA   'auto'
  resolved_bottom           CDATA  #REQUIRED
  resolved_left             CDATA  #REQUIRED
  resolved_right            CDATA  #REQUIRED
  resolved_top              CDATA  #REQUIRED
  resolved_height           CDATA  #REQUIRED
  resolved_width            CDATA  #REQUIRED
  showBackground (always|whenActive) 'always'
  background-color          CDATA  #IMPLIED
  resolved_background-color CDATA  #REQUIRED
  z-index                   CDATA  #IMPLIED
  resolved_z-index CDATA           #IMPLIED
  fit (hidden|fill|meet|scroll|slice) #IMPLIED
  regPoint  CDATA  #IMPLIED
  %regAlign.attr;
>
```

```
<!ELEMENT regPoint EMPTY >
<!ATTLIST regPoint
  name    CDATA  #REQUIRED
  top     CDATA  'auto'
  left    CDATA  'auto'
  right   CDATA  'auto'
  bottom  CDATA  'auto'
  %regAlign.attr;
>
<!-- === timing elements: timing, seq & par
==================================================== -->

<!ELEMENT timing (par | seq | excl) >

<!ELEMENT seq (par | seq |excl)*>
<!ATTLIST seq
        id                          ID              #REQUIRED
        fill (remove|freeze|hold|transition|auto|default) 'default'
        dur                         CDATA           #IMPLIED
        repeatCount                 CDATA           #IMPLIED
        repeatDur                   CDATA           #IMPLIED
        begin                       CDATA           #IMPLIED
        end                         CDATA           #IMPLIED
        repeat                      CDATA           #IMPLIED
        min                         CDATA           '0'
        max                         CDATA           'indefinite'
        restart      (always|whenNotActive|never|default) 'default'
        restartDefault (inherit|always|never|whenNotActive) 'inherit'
        syncBehavior (canSlip|locked|independent|default) 'default'
        syncTolerance       CDATA           'default'
        syncBehaviorDefault (canSlip|locked|independent|inherit)
'inherit'
        syncToleranceDefault CDATA          'inherit'
        fillDefault (remove|freeze|hold|transition|auto|inherit)
'inherit'
        %system.attrs;
        customTest                  CDATA           #IMPLIED
>

<!ELEMENT par (par | seq | excl)*>
<!ATTLIST par
        endsync                     CDATA           'last'
        id                          ID              #REQUIRED
        fill (remove|freeze|hold|transition|auto|default) 'default'
        dur                         CDATA           #IMPLIED
        repeatCount                 CDATA           #IMPLIED
        repeatDur                   CDATA           #IMPLIED
        begin                       CDATA           #IMPLIED
        end                         CDATA           #IMPLIED
        repeat                      CDATA           #IMPLIED
        min                         CDATA           '0'
        max                         CDATA           'indefinite'
        restart (always|whenNotActive|never|default) 'default'
        restartDefault (inherit|always|never|whenNotActive) 'inherit'
        syncBehavior (canSlip|locked|independent|default) 'default'
        syncTolerance CDATA           'default'
```

```
        syncBehaviorDefault (canSlip|locked|independent|inherit)
'inherit'
        syncToleranceDefault        CDATA           'inherit'
        fillDefault (remove|freeze|hold|transition|auto|inherit)
'inherit'
        %system.attrs;
        customTest                  CDATA           #IMPLIED
>

<!ELEMENT excl ((par | seq | excl)* | priorityClass+) >
<!ATTLIST excl
        endsync                     CDATA           'last'
        skip-content                (true|false)    'true'
        id                          ID              #REQUIRED
        fill (remove|freeze|hold|transition|auto|default) 'default'
        dur                         CDATA           #IMPLIED
        repeatCount                 CDATA           #IMPLIED
        repeatDur                   CDATA           #IMPLIED
        begin                       CDATA           #IMPLIED
        end                         CDATA           #IMPLIED
        repeat                      CDATA           #IMPLIED
        min                         CDATA           '0'
        max                         CDATA           'indefinite'
        restart (always|whenNotActive|never|default) 'default'
        restartDefault (inherit|always|never|whenNotActive) 'inherit'
        syncBehavior (canSlip|locked|independent|default) 'default'
        syncTolerance               CDATA           'default'
        syncBehaviorDefault (canSlip|locked|independent|inherit)
'inherit'
        syncToleranceDefault        CDATA           'inherit'
        fillDefault (remove|freeze|hold|transition|auto|inherit)
'inherit'
        %system.attrs;
        customTest                  CDATA           #IMPLIED
>

<!ELEMENT priorityClass EMPTY >

<!-- to be defined more precisely in future versions -->
<!-- === reference elements: references, layout-ref & ref
===================================== -->

<!ELEMENT references layout-ref >

<!ELEMENT layout-ref (ref*) >

<!ELEMENT ref EMPTY >
<!ATTLIST ref
  objectId  CDATA  #REQUIRED
  regionId  CDATA  #IMPLIED
>

<!-- === media elements: media, text, content, img, area
===================================== -->
<!ENTITY % media.type    " image | audio | video | text | animation " >
<!ENTITY % media.attrs
  "id ID #REQUIRED
```

```
  prefer_height CDATA  #REQUIRED
  prefer_width  CDATA  #REQUIRED
  prefer_top    CDATA  #REQUIRED
  prefer_bottom CDATA  #REQUIRED
  actual_type   %media.type;  #REQUIRED
  fit           (hidden|fill|meet|scroll|slice)  #REQUIRED
  src           CDATA  #REQUIRED"
>

<!ELEMENT media (text | img | video | audio | animation)* >

<!ELEMENT text content? >
<!ATTLIST text %media.attrs; >

<!ELEMENT img (area*) >
<!ATTLIST img %media.attrs; >

<!ELEMENT audio (area*) >
<!ATTLIST audio %media.attrs; >

<!ELEMENT video (area*) >
<!ATTLIST video %media.attrs; >

<!ELEMENT animation (area*) >
<!ATTLIST animation %media.attrs; >

<!ELEMENT area EMPTY >
<!ATTLIST area
            href  CDATA  #REQUIRED
        begin                       CDATA          #IMPLIED
        end                         CDATA          #IMPLIED
        dur                         CDATA          #IMPLIED
        repeatCount                 CDATA          #IMPLIED
        repeatDur                   CDATA          #IMPLIED
        repeat                      CDATA          #IMPLIED
        min                         CDATA          '0'
        max                         CDATA          'indefinite'
        %system.attrs;
        customTest                  CDATA          #IMPLIED
>
```