Université Joseph Fourier

U.F.R Informatique &
Mathématiques Appliquées

Institut National
Polytechnique de Grenoble

ENSIMAG

I.M.A.G.

ÉCOLE DOCTORALE
MATHÉMATIQUES ET INFORMATIQUE

DEA INFORMATIQUE :
SYSTÈMES ET COMMUNICATION

Projet présenté par :

Fateh Boulmaiz

COUPLAGE D'UN LANGAGE DE CONTRÔLE DE FORMATAGE AVEC UN SYSTÈME DE FORMATAGE EXISTANT

Effectué au laboratoire de l'Institut National de Recherche en Informatique et en Automatique -INRIA
Dans le cadre du projet WAM

23 Juin 2003

Jury :

Joëlle	Coutaz
Jacques	Briat
Jérôme	Gensel
Cécile	Roisin
Frederic	Bes

Résumé

Le travail présenté dans ce rapport a pour objectif de contribuer au domaine de la présentation de *documents multimédia*. Nous nous intérressons tout particulièrement aux problèmes de conception des formateurs XML. En considérant le problème de la composition des formateurs, nous initialisons une nouvelle direction de recherche sur la conception des formateurs XML par *composition*.

La complexité croissante des documents multimédia, la multiplication des environnements d'exécution et la diversification des attentes des utilisateurs et des auteurs de ces documents ont conduit à l'émergence de nombreux langages de présentation et d'outils de formatage associés (CSS, SVG, SMIL, XSL-FO). Malgré cette évolution vers des systèmes plus puissants et plus performants, les langages comme leurs formateurs sont restés très influencés par les méthodes de formatage de texte. Ils restent très rigides dans leur fonctionnement et donnent peu de solutions face aux nouveaux besoins actuels des documents multimédia.

Depuis quelques temps, la communauté de recherche tente de combler ce fossé et a proposé, parmi d'autres, d'utiliser de nouvelles techniques à base de contraintes pour permettre d'étendre l'expressivité des langages de présentation. Cette approche offre une amélioration très intéressante du pouvoir d'expression offert aux auteurs mais les formateurs associés deviennent de plus en plus complexes et difficiles à mettre en ouvre, et donc à étendre dans le futur.

Dans ce stage, nous proposons une architecture logicielle pour un système de présentation multimédia qui permet le couplage d'un langage de présentation et des services de formatage associés avec des systèmes de formatage existants de langage standard comme SMIL ou XSL-FO. Nous voulons non seulement être capable d'intégrer ce langage dans les langages existants mais également de proposer des services de formatage visant à intégrer son traitement dans les formateurs existants. Dans ce cas, le formateur est obtenu par composition (assemblage) des formateurs des langages existants. Donc, Notre ambition est double : intégration des langages et intégration des traitements. Le but est de permettre une meilleure réutilisation des capacités de contrôle offertes par ces langages ainsi que de leurs formateurs.

Mots clés

XML. Formatage de document. Langages de présentation : SMIL, XSL-FO, CSS,... Formateurs XML. Architecture logicielle. Modèles à composant.

Remerciements

Que tous ceux qui m'ont aidé trouve ici toute ma reconnaissance et ma gratitude la plus sincère.

Je remercie Vincent Quint, chef du projet WAM, pour m'avoir accueilli au sein de l'équipe WAM ;

J'exprime toute ma reconnaissance à mes encadreurs Cécile Roisin et Frederic Bes sans lesquels je n'aurais très certainement pas pu terminer ce stage ;

Je remercie Joëlle Coutaz et Jacques Briat pour avoir voulu évaluer ce travail ;

Je remercie Jérôme Gensel qui m'a honoré en acceptant d'être membre du jury ;

Je remercie Nabil Layaïda, Tayeb Lemlouma et Daniel Weck pour leur aide et leurs conseils ;

Toute ma gratitude à toutes les personnes ayant relu et commenté ce manuscrit ;

Je salue tous les membres de l'équipe WAM pour leur gentillesse et la générosité de leur acceuil ;

Et enfin, je tiens à avoir une pensée toute particulière à ma famille et surtout mes parents.

Table des matières

Table des figures

Introduction

Les nouvelles technologies rencontrent un grand engouement tant dans la vie professionnelle que domestique. Le domaine des documents électroniques n'y échappe pas. Notamment par le fait que la communication se fait de plus en plus en utilisant des supports qui intègrent du son, de la vidéo, de l'animation et du texte.

Le développement des réseaux de communication rapides avec en particulier l'explosion de l'Internet, la complexité croissante des applications, le déploiement de grandes variétés de terminaux et la diversification des attentes des utilisateurs lancent de nouveaux défis à la recherche dans le domaine de traitement des documents électroniques. En réponse à ces attentes, ce domaine a beaucoup évolué en quelques années pour passer du traitement d'informations purement textuelles à des informations structurées, hypermédia ou encore multimédia.

Un *document multimédia* permet de combiner des objets de différents types comme le texte, les images, l'audio, et la vidéo à l'intérieur d'un même document électronique en tenant compte de contraintes de synchronisation inhérentes à l'aspect temporel de ces objets.

Des efforts ont porté et continuent de porter sur la fourniture de solutions pour la création et pour la présentation des documents multimédia. De nombreux logiciels permettent de créer de tels documents de façon *ad hoc.* Néanmoins - avec toutes les limites que recèlent ces logiciels, notamment la création des applications dans un format propriétaire – il n'est pas préférable d'utiliser ces logiciels pour la création des documents multimédia.

La définition de formats standard est bien sûr la solution " idéale " pour répondre aux besoins de portabilité, d'échange, de pouvoir d'expression, d'interrogation et de recherche d'information et enfin de visualisation et de présentation des documents multimédia. Ainsi, les groupes du consortium W3C ont adopté le standard XML, qui peut être considéré comme le standard pivot du Web à partir du quel et autour duquel les autres standards de représentation des informations du Web sont définis. XML est un standard qui permet la modélisation des documents.

Un *modèle de document* regroupe les documents qui obéissent à un même ensemble de contraintes.

Un des apports majeurs de XML est la séparation de contenu du document de sa présentation. Or, il faut fournir des outils pour permettre la perception de ce contenu sur des terminaux aussi divers que des PC, des PDA (Portable

Digital Assistant- assistants électroniques de poche) ou même des téléphones cellulaires.

Le contexte dans le quel se situe ce stage est marqué par l'apparition d'une offre abondante de traitements portant sur la présentation des documents multimédia, la maturité de ces travaux est couronnée par l'émergence de nombreux langages de présentation et d'outils de formatage associés. C'est ainsi que, parmi d'autres, les standards SMIL, XSL-FO, CSS, SVG et MathML ont été proposés.

Ces langages n'offrent pas le même niveau de souplesse ni le même pouvoir d'expression. Cependant, une limite partagée par ces langages est qu'ils couvrent seulement en partie les besoins de présentation des documents multimédia. Ils permettent de spécifier des présentations relativement rigides. En guise d'exemple, le langage SMIL ne permet que la spécification des placements absolus. Il n'est pas possible de spécifier des placements relatifs comme par exemple : l'objet A est à droite de l'objet B.

Cette rigidité entraîne souvent une baisse de la qualité de la forme finale (la présentation) des documents. En particulier, les formateurs entrent en *situation d'échec*, soit parce qu'ils ne peuvent trouver de solution satisfaisant le jeu de contraintes de la spécification de l'auteur, soit que la solution fournie n'est pas celle qui est attendue.

Ce manque de souplesse a motivé certains travaux de recherche – tels ceux qui utilisent des techniques à base de contraintes - qui visent à étendre l'expressivité de ces langages. La finalité de ces travaux est d'utiliser un langage laissant plus de souplesse dans la spécification pour que le formateur ait à sa disposition un espace de solutions dans lequel il calculera la solution adaptée à chaque contexte. Dans cette catégorie, nous citons Madeus et Cuypers.

Néanmoins, les expériences les plus probantes dans ce domaine montrent que même cette technique n'offre pas de solution tout à fait satisfaisante. Cette frustration du résultat de formatage n'est pas seulement due au manque de maîtrise des outils par l'auteur, mais aussi à la complexité grandissante des fonctions que l'on souhaite faire réaliser par les formateurs. En effet, jusqu'à présent, la démarche adoptée pour combler les lacunes des langages de présentation était de spécifier un nouveau langage de présentation " à partir de zéro " et par conséquent, écrire un nouveau formateur spécifique dont le code augmente proportionnellement en complexité et en taille. Plus le code de ces applications grossit plus leur maintenance devient difficile. Que ce soit pour des corrections de bugs, des ajouts de fonctionnalités ou des modifications, la réécriture du code est ardue et longue.

En général, les techniques à base de contraintes sont très difficiles à mettre en œuvre et à gérer. Ce point évoque une question fondamentale qui a motivé et qui continue à motiver des recherches en génie logiciel, celle de l'architecture logicielle des systèmes informatiques complexes. En effet, si le choix de l'architecture est approprié, cela permet au produit de satisfaire les exigences et d'être facilement modifiable. Au contraire, une architecture mal pensée peut poser un certain nombre de problèmes notamment lorsque des modifications doivent être faites au système.

Pour répondre à ces problèmes, une nouvelle démarche pour améliorer le processus de formatage est de proposer des solutions de formatage pour compléter

les formateurs actuels plutôt que de les remplacer. C'est dans cet esprit que l'équipe WAM travaille sur la réalisation d'un service de formatage XEF, qui intègre des éléments permettant aux auteurs de mieux contrôler le formatage de leurs documents multimédia : utilisation de priorité, de propriétés globales et de stratégies de repli. Ce langage a donné lieu à l'élaboration d'un formateur servant à expérimenter son expressivité et ses capacités de contrôle.

Notre objectif est de proposer une architecture permettant le couplage du langage XEF et des services de formatage associés avec des systèmes de formatage existants comme SMIL, Madeus et CSS. Notre ambition dans ce stage est double : intégration du langage et intégration du traitement. Le travail demandé s'inscrit donc à deux niveaux différents puisqu'il concerne d'une part le couplage au niveau des langages et d'autre part celui au niveau des traitements (les formateurs). Il conviendra donc d'étudier et de proposer des solutions cohérentes entre ces deux niveaux.

Il est clair que les solutions à ces deux problèmes ne sont pas autonomes mais bien au contraire interdépendantes. En effet, l'efficacité du couplage au niveau des traitements repose principalement sur la technique de couplage adoptée au niveau des langages. Inversement, un couplage au niveau de l'architecture logicielle peut imposer un mode de couplage au niveau des langages.

La solution à proposer pour répondre au premier besoin devra tenir compte du fait que certains éléments pourront être exprimés dans les deux langages à la fois. Ce point évoque notamment les problèmes de cohérence qui peuvent survenir entre les deux spécifications. En outre, la solution doit permettre à l'auteur de contrôler des documents nouveaux aussi bien que des documents existants à un moindre coût.

Au niveau des traitements, l'architecture aura comme challenge de proposer un système complet reprenant tous les aspects de chaque langage. La difficulté est d'arriver à tirer au maximum parti des traitements existants aussi bien d'un côté que l'autre, sans avoir à les réécrire. L'influence du langage de présentation à contrôler est à étudier pour voir si une architecture générique est possible.

Des techniques comme la programmation à base de composants, ont été proposées, dans d'autres contextes, pour répondre aux besoins cités ci-dessus. Afin de pouvoir appliquer ces techniques dans le contexte des langages de présentation basés sur XML, il faut identifier et définir les besoins relevés par notre problématique. Ce travail recouvre les thèmes suivants :

- Définir les termes documents et documents composites,
- Modéliser le processus de formatage des documents,
- Explorer les différentes techniques existantes de couplage des langages basés sur XML et des formateurs associés,
- Concevoir une architecture logicielle pour un système de formateurs XML extensible et adaptable.

Bien qu'alimentée par un contexte spécifique, celui du couplage du langage XEF avec des langages de présentation existants, notre problématique se veut plus large que l'étude de ce cas particulier. Nous nous intéressons, plus généralement, à la conception d'un système permettant la création et la manipulation des documents composites par la composition des langages de balisage et des formateurs existants.

L'intérêt de notre proposition est triple. Tout d'abord, cette approche permet à l'auteur non seulement de contrôler le formatage des nouveaux documents mais de pouvoir contrôler le formatage des documents existants sans qu'il ait besoin de modifier le document source. Ensuite, réduire le travail du développeur de l'application. En effet, au lieu d'étendre le formateur associé au langage à contrôler en codant directement les nouvelles fonctionnalités, il suffit de coupler les formateurs des deux langages. Enfin, un avantage majeur de cette démarche, d'ailleurs la principale motivation de ces travaux, est celui d'arriver à mieux contrôler le formatage en tirant profit des avantages offerts par chaque formateur.

Enfin, nous nous sommes fixés comme objectif de faciliter le processus de couplage des formateurs. Il s'agit en particulier de définir un langage de composition permettant d'assembler facilement des formateurs.

Ce stage a été effectué au sein du projet WAM – Web, Adaptation et Multimédia – du laboratoire INRIA Rhône-Alpes. Ce projet fait suite au projet OPERA.

Le projet WAM aborde quelques problèmes posés par les évolutions du Web. Il se focalise sur la *transformation* de documents considérée comme un type de traitement générique des documents du Web. Il considère plus particulièrement les *documents multimédia* qui intègrent étroitement des média statiques (texte, images, équations) et dynamiques (vidéo, son, animations). Il applique ses résultats en particulier à *l'adaptation* des documents et à l'indépendance des appareils d'accès au Web.

Plan du mémoire

Ce mémoire est organisé en deux parties. Dans une première partie, nous dressons un état de l'art sur le traitement des documents multimédia, les techniques de couplage des langages basés sur XML et les techniques de couplage des architectures logicielles. Dans une seconde partie nous présentons notre proposition : une architecture générique pour le couplage d'un langage de contrôle de formatage avec un système de formatage existant. Enfin, nous concluons et présentons les perspectives de notre travail. Nous donnons ci-dessous le résumé de chaque chapitre.

Partie 1 : Etat de l'art

Chapitre 1 : Les documents multimédia

Dans ce chapitre, nous introduisons les notions de base sur les documents multimédia. Le but est de familiariser le lecteur à ce domaine en donnant les définitions et en exposant les principes du processus de création d'un document multimédia. En particulier, nous présentons le processus de formatage qui est au cœur de notre problématique. Nous identifions les besoins liés à ce domaine et les limites actuelles.

Chapitre 2 : Couplage des langages basés sur XML

Ce chapitre est consacré à l'étude des techniques de couplage des langages basés sur XML. Pour chaque technique, nous présentons le principe et nous discutons ses avantages et ses limites.

Chapitre 3 : Couplage des architectures logicielles

Dans ce chapitre, nous étudions les travaux émergents concernant les architectures logicielles. Nous verrons que l'accent a été mis sur la propriété de réutilisation des codes existants. Nous avons découpé cette étude en trois parties, la première est un état de l'art sur la composition et le couplage des architectures logicielles. La deuxième partie porte sur l'étude de la réutilisation dans la technologie à composant. Enfin, la dernière partie concerne le couplage des formateurs XML.

Partie 2 : Contribution

Chapitre 4 : Une architecture pour le couplage de formateurs

Le but de ce chapitre est de donner un aperçu sur la solution afin de faciliter la lecture de la suite de ce rapport. Nous commençons ce chapitre par une présentation de notre contexte de travail qui est le langage XEF, un langage de contrôle de formatage défini dans le projet WAM. Ensuite, nous décrivons brièvement notre proposition d'un système permettant de faire le couplage d'un langage de contrôle de formatage avec un système de formatage existant.

Chapitre 5 : Couplage du langage XEF avec un langage de formatage existant

Dans ce chapitre, nous décrivons notre proposition pour le couplage d'un langage de contrôle de formatage avec un système de formatage existant. Nous illustrons nos propos à travers un document spécifié dans le langage SMIL. La démarche suivie dans ce chapitre consiste à présenter plusieurs techniques de couplage dont on discute les avantages et les limites de chaque technique par rapport aux besoins de couplage d'un langage de contrôle de formatage avec un langage de présentation.

Chapitre 6 : Une architecture logicielle pour le couplage du formateur XEF avec un formateur existant

Dans ce chapitre, nous abordons le problème du couplage des formateurs XML. Une attention particulière est accordée à l'interaction entre les formateurs. Nous proposons une architecture à couplage faible entre les formateurs.

Chapitre 7 : Mise en œuvre et tests

Dans ce chapitre, nous présentons une mise en œuvre de notre proposition. L'objectif de ce chapitre est double : d'une part, évaluer l'architecture que nous proposons par rapport à nos objectifs et d'autre part, valider le langage XEF.

Chapitre 8 : Conclusion

Dans ce chapitre, nous résumons l'apport de ce travail et nous donnons des perspectives pour des travaux futurs.

Première partie

État de l'art

Chapitre 1

Les documents multimédia

1.1 Introduction

Nous nous intéressons dans ce premier chapitre aux documents multimédia. En effet, ces dernières années, le contenu des documents électroniques a beaucoup évolué et il est devenu de plus en plus riche en intégrant des objets de natures différentes comme la vidéo, les images, le son, ou encore les animations. Ces objets sont organisés dans ce qu'on appelle des documents multimédia dont le rôle est d'exprimer des relations spatio-temporelles entre ces objets.

La motivation de ce premier chapitre est double :

- D'une part familiariser le lecteur aux documents électroniques et en particulier les documents multimédia, un thème récurrent dans la thématique de ce stage. Pour cela et après avoir donné quelques définitions que nous jugeons nécessaires pour faciliter la lecture de ce rapport, nous présentons succinctement la technologie XML et ses applications. En effet, depuis de nombreuses années, les documents électroniques ont fait l'objet d'études qui ont conduit à l'identification de caractéristiques attachées aux documents et classées selon différentes dimensions. Le résultat majeur de ces travaux est la définition de standards comme XML qui permettent de modéliser ces documents. Aujourd'hui, dés que l'on parle de documents multimédia, la notion de XML apparaît. Il nous semble donc important de faire un tour d'horizon de cette technologie.
- D'autre part, positionner le contexte de ce projet par rapport aux problématiques liées aux documents multimédia. En effet, de nombreux travaux dans le domaine du génie documentaire ont permis de repérer certains problèmes comme : la modélisation, la transformation, l'adaptation et le formatage. Bien que ces problèmes ne soient pas autonomes, nous allons uniquement considérer le problème de formatage.

1.2 Documents multimédia

1.2.1 Définitions

Il existe, selon le contexte, une myriade de définitions d'un document. Dans le domaine informatique, nous empruntons à [Vil02] la définition suivante :

Un *document désigne un ensemble cohérent et fini, d'informations plus au moins structurées, perceptibles, à usage défini et représentées sur un support concret.*

Cette définition assez générale rend compte à la fois du type d'informations manipulées et de leur perceptivité. Ce dernier point concerne par exemple la représentation graphique de ces informations sur un support de visualisation comme un PC, une imprimante ou un téléphone cellulaire.

Les documents peuvent être classés en fonction de plusieurs critères. Une première classification est faite selon l'organisation des éléments de base. La gamme des documents ainsi considérés s'étend depuis les documents composés d'une liste d'éléments de base sans aucune structure jusqu'aux documents fortement structurés. Un document est structuré s'il est basé sur une organisation logique de ses éléments [Car97] (cf.§.1.2.2).

Selon la nature des éléments de base qui les composent, les documents peuvent être classés en plusieurs types : documents textuels, documents images, documents vidéo ou encore des documents multimédia. Ces derniers rencontrent, récemment, un vif engouement dans notre société. Notamment avec l'expansion fulgurante du Web et l'arrivé de PC dits multimédia. C'est ainsi que le domaine de traitement des documents a évolué pour passer du traitement des documents textuels aux documents multimédia.

Nous consacrons le reste de cette section à l'introduction des notions de base des documents multimédia. Nous commençons par préciser le sens que nous accordons aux documents multimédia. Ensuite, nous présentons le processus de génération d'un document multimédia. En particulier, nous faisons le point sur le processus de formatage.

Un *document multimédia* est un document interactif, temporisé et contenants des éléments de nature diverses : texte, image, audio, vidéo,...

Comme l'indique la définition, l'un des caractéristiques des documents multimédia est l'interactivité. Mieux encore, cette définition met l'accent sur l'aspect temporel des documents multimédia inhérent à l'intégration de données temporisées comme les éléments vidéo et sonores.

1.2.2 Modèle de document multimédia

Un modèle de document permet de représenter toutes les relations qui peuvent exister entre ses éléments. Ces relations sont de quatre types [Lay97] :

- *La dimension logique* permet de regrouper des éléments qui sont liés sémantiquement. Par exemple, un chapitre est constitué d'une introduction et d'un ensemble de sections,
- *La dimension spatiale* concerne l'affectation des objets dans l'espace,
- *La dimension hypermédia* permet de définir des relations sémantiques

entre les éléments ou plus généralement entre des documents,
- *La dimension temporelle* est la caractéristique majeure d'un document multimédia. Elle définit le scénario temporel du document, c'est-à-dire, l'organisation des éléments dans le temps.

1.2.3 Processus de présentation d'un document multimédia

1.2.3.1 Besoins

Le processus de génération d'un document multimédia exprime un certain nombre de besoins spécifiques. Ces besoins sont d'ores et déjà identifiés dans [RV] :

- *Les besoins liés à la conception* : le processus de conception des présentations multimédia est complexe. Cela est inhérent à la nature même des objets média et la façon dont ils sont composés (composition spatiale, temporelle,...). L'outil de génération doit permettre à l'auteur d'exprimer ces relations de composition ainsi que la perception des caractéristiques de ces objets,
- *Les besoins d'expression* : la variété des types de média utilisés dans un document multimédia permettent aux auteurs de jouer sur plusieurs facteurs : l'organisation temporelle, l'organisation spatiale, ...Ainsi, il est nécessaire d'offrir aux auteurs les moyens d'utiliser cette richesse que se soit en terme de média de base que d'organisation spatiale et temporelle de ces média,
- *Les besoins liés à la présentation* : outre les services nécessaires à la restitution des information multimédia (accès aux média, synchronisation spatiale et temporelle, ...), de nombreuses applications souhaitent obtenir différentes présentations pour des documents de même classe ou pour un même document. En effet, la multiplicité des terminaux (PDA, téléphone cellulaire, PC,...) et le contexte de présentation (par exemple la modélisation de l'utilisateur) rend nécessaire la production de plusieurs types de présentation à partir d'une même source d'informations.

1.2.3.2 Outils de création

Suivant l'outil d'édition utilisé, le processus de génération d'un document multimédia peut se faire selon trois approches [Thu03]. La génération en utilisant un langage de programmation, des outils propriétaires ou le format textuel.

- *Les langages de programmation* : dans cette approche, l'auteur décrit la présentation multimédia en utilisant un langage de programmation impératif de type Java et C. C'est notamment le cas de *Encarta* et de *Atlas mondial*. Ces langages offrent des bibliothèques – comme *Java Media Framework* en Java – qui permettent de manipuler les différents types de média existants. L'inconvénient majeur de cette approche est le coût relativement élevé de développement. En particulier, cette approche requiert des compétences en programmation. Ce qui est contradictoire avec le but du génie documentaire qui vise à simplifier la tâche de création des documents multimédia.

– *Les outils propriétaires* : dans le but de réduire le coût de génération des présentations multimédia, un grand nombre d'entreprises ont sorti leur propre solution. *PowerPoint* de MicroSoft, *Flash* de Macromédia et *Adobe Studio* de Adobe, pour ne citer que quelques exemples.
Toutes ces solutions ont en commun l'objectif de rendre la tâche de création des présentations multimédia plus facile. Ils n'ont cependant pas les mêmes possibilités de description et n'offrent pas les mêmes outils.
L'inconvénient majeur de toutes ces applications est que les présentations multimédia sont générées dans un format propriétaire et binaire. Avec tous les inconvénients des formats propriétaires : elles limitent très fortement les possibilités d'échange de données avec d'autres applications. En plus, les formats propriétaires ne sont pas manipulables en toute liberté par des utilisateurs qui ne disposent pas des outils payants.
Une autre limite de ces formats vient du fait que les objets manipulés sont des boites noires. Il n'est pas possible d'en extraire des informations sémantiques. Or, il est difficile de mettre en œuvre des techniques de recherche d'information.
Il est important, dans la mesure du possible, de bannir les formats propriétaires et de leur préférer des formats libres de tous droits.
– *Les formats textuels* : des recommandations visant l'amélioration et l'ajout de données ont été formulées dans le but de combler les lacunes décelées dans la technique précédente. Il a été recommandé de mettre particulièrement l'accent sur deux points : la déclarativité et l'indépendance vis-à-vis l'outil de génération. Ainsi, un nouveau paradigme dans le domaine du génie documentaire est apparu. L'idée clé est d'utiliser un format textuel pour décrire la structure et le scénario temporel du document multimédia. Dans ce cadre plusieurs modèles ont été proposés, nous décrivons dans le paragraphe (§.1.3.1) quelques modèles qui permettent cette description textuelle.
Un intérêt majeur de cette approche est qu'elle permet non seulement la séparation des données et des traitements, mais surtout leur indépendance.

1.2.3.3 Processus de présentation

De nombreux travaux ont été faits sur la modélisation du processus de présentation. Cependant la plupart de ces travaux se focalisent sur un aspect de traitement de documents sans les autres. Ainsi on trouve dans la littérature traitant les documents multimédia, des travaux qui considèrent le problème de la conception et de la présentation d'informations multimédia à partir de sources [Vil02], d'autres travaux s'intéressent aux aspects liés aux problèmes de la génération automatique et à l'adaptabilité des documents multimédia et enfin des travaux qui portent sur la description sémantique des médias [Thu03] pour l'indexation et la recherche d'information.

Nous tenons à rester très général dans la présentation du processus de présentation des documents multimédia. Pour ce faire, nous nous servons du modèle proposé dans [RV]. Dans ce modèle, le processus de présentation (cf.-FIG.1.1) prend en entrée un document source XML quelconque et produit en sortie un document multimédia. La présentation d'un document multimédia

s'effectue en deux étapes :

- *Etape de transformation* : qui permet de créer les objets multimédia à partir du document source. Elle permet aussi de les synchroniser (dimension temporelle), les positionner (dimension spatiale) et de les lier (dimension hypermédia). Le résultat de cette étape est un document multimédia décrit dans un langage de présentation (cf.§.1.3.2.4),
- *Etape de formatage* : qui permet de générer un document directement interprétable par le système de présentation (cf.§.1.3.2).

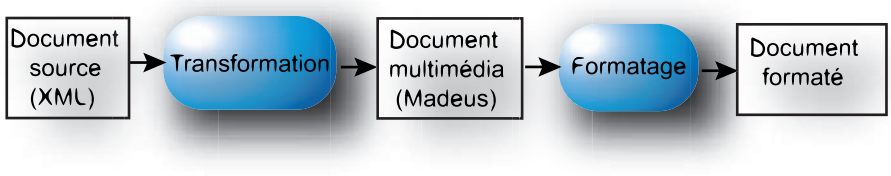

FIG. 1.1 – Processus de présentation d'un document multimédia

On peut toujours se passer de la première étape en spécifiant la présentation directement dans un langage de présentation.

1.3 Technologie XML

1.3.1 Préambule

Dans l'approche déclarative, plusieurs modèles ont été proposés pour définir la structure logique du document et exprimer les relations spatio-temporelles entre ses objets. La première solution est apparue avec le standard SGML[1].

SGML [SGML86] est une norme internationale pour l'échange de documents structurés. Il s'agit d'un langage, développé spécifiquement dans les années 80 pour la représentation formelle d'un document. Une caracrtéristique intéressante de SGML est la *séparation entre le contenu d'un document et sa présentation.*

Bien qu'utilisé dans certains domaines de l'industrie pour réaliser de grandes documentations techniques comme dans l'industrie aéronautique, il semble souffrir de sa rigidité en impliquant une mise en oeuvre complexe, ce qui l'handicape pour une utilisation à grande échelle, notamment sur Internet. SGML ne permettrait pas non plus la mise en oeuvre d'un hypertexte ouvert.

C'est d'ailleurs sa complexité et son coût de mise en oeuvre élevé qui ont motivé le développement de solutions de remplacement.

Pour faire face à cette complexité, un nouveau standard est proposé. Il s'agit de HTML [RLJ99]. Ce dernier est devenu le langage de marquage le plus répandu pour les documents basés Web. L'augmentation de sa popularité a également

[1]Standard Generalized Markup Language.

révélé ses limites. Ces dernières concernent notamment le nombre réduit d'ensembles de balises que l'utilisateur peut utiliser. Les auteurs du langage HTML ne peuvent pas créer leurs propres balises car les navigateurs Web disponibles ne connaissent que celles appartenant aux standards HTML supportés. Autre limite du langage HTML : il va plutôt à l'encontre de la philosophie SGML qui vise à séparer le balisage du contenu d'un document du balisage de son apparence. Il est pensé principalement pour la présentation et il ne peut servir à l'échange de données.

Une solution était de se rapprocher du langage SGML, tout en gardant la simplicité du langage HTML : il s'agit du standard XML.

XML [BPMM00], acronyme de langage extensible de marquage[2] est une réponse à un besoin de diffusion de données à grande échelle, rendu particulièrement important par le développement d'environnements distribués et notamment par Internet. Il est alors indispensable de disposer d'une représentation structurée des données. Plus précisément, l'idée directrice est de réaliser des documents auto-descriptifs en représentant dans le document la sémantique des données qui y figurent. Les applications sont alors capables d'exploiter cette sémantique.

L'ensemble de ces caractéristiques fait en sorte que la norme XML permet de représenter des documents structurés, elle assure la pérennité de l'information et facilite les échanges de données.

Un modèle de document XML peut être spécifié à l'aide d'un schèma comme XML Schema [TBMM01] et DTD [BPMM00].

1.3.2 Formatage

Dans le paragraphe précédent, nous avons présenté quelques modèles permettant la modélisation des documents multimédia. Nous avons souligné en particulier que le point est mis sur la séparation du contenu de ces documents de leur présentation. Il est nécessaire de se disposer des moyens qui permettent la perception de ces données.

Nous consacrons le reste de ce chapitre à l'étude du processus qui permet cette perception, à savoir le formatage.

1.3.2.1 Définition

Le formatage est le processus qui consiste à convertir des informations de présentation vers des informations concrètes [Qui87][3].

1.3.2.2 Besoins

Compte tenu de la discussion précédente, un certain nombre des qualités requises pour un langage de présentation sont relevé dans [SJ]et [RHA95]. Parmi celles-ci nous retenons :

[2]XML (eXtensible Markup Language), est une recommandation du consortium W3C depuis février 1998.

[3]Cité dans [Vil02].

– *Effort proportionnel* : le langage de présentation doit permettre la définition des présentations complexes mais la spécification des présentations simples doit rester une tâche facile,
– *Déclarativité* : en langages déclaratifs, les présentations sont lisibles, compréhensibles et le concepteur n'a pas besoin de connaître l'ordre d'exécution des instructions.

1.3.2.3 Techniques de formatage

Différentes approches de formatage existent, la différence étant due à la façon de calculer les valeurs des propriétés [Vil02] :

– La première approche est la plus simple à mettre en œuvre et consiste à calculer les valeurs absolues des propriétés. Par exemple la conversion de la valeur d'une couleur en une valeur directement interprétée par le système d'exécution,
– La deuxième approche fait appel au calcul hiérarchique sans cycle. Dans CSS par exemple, le calcul des valeurs héritées des propriétés comme la couleur et la taille des caractères utilise cette technique,
– La troisième approche, utilise les techniques du calcul hiérarchique avec cycle. En fonction de degré de flexibilité voulu, les algorithmes de formatage sont plus en moins complexe. Ainsi, le langage Madeus (cf.§.1.3.2.4.3) qui permet une certaine souplesse dans la spécification des relations spatio-temporelles entre les média nécessite l'utilisation des résolveurs de contraintes[4] pour le formatage des documents.

1.3.2.4 Quelques langages de formatage

Nous étudions dans ce paragraphe quelques langages de présentation. La finalité de cette étude, qui est loin d'être exhaustive ou complète, est de montrer les lacunes des modèles de présentation actuels plutôt que de présenter la syntaxe de ces modèles.

1.3.2.4.1 SVG

SVG (Scalable Vector Graphics) [SVG1] est une recommandation du consortium W3C qui vise la définition de graphiques vectoriels en 2 dimensions au moyen d'une syntaxe XML. La norme spécifie la description et l'intégration de trois types d'objets :

– Des objets vectoriels 2D,
– Des images,
– Des objets de type textuel.

La norme SVG ne se limite pas à la définition d'objets statiques ; elle inclut également des possibilités d'animation et d'interaction, que ce soit de manière procédurale via l'utilisation de langages de scripts ou de manière déclarative à l'aide du langage proposé par SVG. Ce dernier incorpore les fonctions d'animation définies dans la spécification *SMIL Animation* définis dans SMIL 2.0 (cf.§.1.3.2.4.2).

[4] Nous ne détaillons pas la technique des contraintes. Le lecteur peut se référer à [Tar00].

Dans l'exemple de la figure FIG.1.2, les valeurs de l'animation *viewBox* changent en fonction du temps. Cela donne l'impression d'un zoom vers l'avant sur le document SVG, suivi d'un travelling horizontal et enfin d'un zoom arrière.

```
1. <svg viewBox="0 0 320 200">
2.  <animate attributeName="viewBox" values="0 0 720 800; 50 50 100
    100; 150 50 200 100 0 0 720 800" dur="8s" repeatDur="indefinite"/>
3.  <rect x="10" y="20" width="100" height="200" style="fill :red ;stroke :-
    blue ;stroke-width :4"/>
4.  <circle style="fill :red ; stroke :blue ; stroke-width :4" cx="250" cy="160"
    r="90"/>
5. < /svg>
```

FIG. 1.2 – Exemple de document SVG

Une limite de SVG -et de la majorité des langages de présentation- est qu'il ne permet pas de présenter le texte sur plusieurs lignes. Il n'effectue pas de retour à la ligne ou de césure automatique. C'est à l'auteur ou au logiciel d'édition à calculer les retours à la ligne.

1.3.2.4.2 SMIL

SMIL [SMIL2] est le Langage d'Intégration Multimédias Synchronisés proposé par le consortium W3C dont l'objectif est de pouvoir synchroniser au sein d'une même présentation des média des types différents tels que des images, du son, des vidéos, du texte,...

Le langage SMIL permet de décrire l'organisation spatiale ainsi que l'organisation temporelle des média. L'organisation spatiale est décrite grâce aux éléments : *root-layout*, *layout* et *region*.

Dans l'exemple de la figure FIG.1.3 extrait du document SMIL *ProgrammeCinémas* donné dans le chapitre 5, on spécifie l'organisation spatiale de la présentation SMIL. Celle-ci contient un élément *layout* composé de quatre éléments *region*. Ces éléments décrivent l'organisation spatiale présentée dans la figure FIG.1.4.

Bien qu'il permette d'imbriquer les éléments de l'organisation spatiale - en utilisant les fonctionnalités du module *HierarchicalLayout* - pour positionner les objets de façon relative[5], SMIL n'offre aucun opérateur pour gérer ce positionnement.

Pour l'organisation temporelle, SMIL propose un ensemble d'opérateurs permettant une synchronisation à gros grain des objets (cf.lignes 12-31 de l'exemple donné dans le paragraphe §.5.2.1) :

- L'élément *seq* : joue les éléments fils en séquence,
- L'élément *par* : joue les éléments en parallèle. Il dispose d'un attribut *endsync* qui permet de synchroniser la fin de cet élément avec un de ces

[5]Il s'agit de la recommandation 2.0 du SMIL.

```
1. <layout>
2. <root-layout background="Titanic003.jpg" height="442" width=-
   "770"/>
3. <region id="town" top="15" left="200" height="45"
   width="370"/>
4. <region id="title" top="392" left="200" height="35"
   width="370"/>
5. <region id="video" top="80" left="20" height="292"
   width="350"/>
6. <region id="description" top="80" left="390" height="292" width-
   ="385"/>
7. < /layout>
```

FIG. 1.3 – Exemple de document SMIL

fils. Cet attribut peut prendre la valeur *first* pour indiquer que la fin de *par* intervient avec le premier fils qui se termine mais aussi la valeur *last* pour se synchroniser avec le fils le plus long ou encore faire référence à un élément précis (via son identificateur),

– L'élément *excl* : joue les éléments de façon exclusive (un seul élément est joué).

Mieux encore, SMIL définit des attributs permettant la synchronisation des éléments de façon relativement fine. Ainsi les attributs *begin* et *end* attachés aux média et aux opérateurs temporels permettent de spécifier des arcs de synchronisation par événements (un élément se joue deux secondes après le début d'un autre,...). SMIL offre aussi un élément *switch* permettant de spécifier un ensemble d'éléments alternatifs dont seul le premier élément alternatif acceptable est choisi. Ce choix est effectué par rapport à la valeur des attributs de test prédéfinis mais, pour permettre plus de flexibilité dans la sélection, SMIL 2.0 permet la définition d'attributs de test personnalisés définis par l'auteur.

SMIL 2.0 est conçu dans le but de permettre de réutiliser la syntaxe et la sémantique de SMIL dans d'autres langages basés sur XML, en particulier ceux qui nécessitent de représenter une temporisation et une synchronisation. Par exemple, les composants de SMIL 2.0 sont utilisés pour intégrer la temporisation dans XHTML [XHTML10] et dans SVG [SVG1]. La stratégie prise en compte pour permettre cette réutilisation est fondée sur le concept de modularisation.

La modularisation est une approche selon laquelle une fonctionnalité de balisage est spécifiée comme un ensemble de modules qui contiennent des éléments, des attributs et des valeurs d'attributs sémantiquement liés à XML. Par exemple, les modules d'animation de SMIL 2.0, qui sont composés d'un module *BasicAnimation* et d'un module *SplineAnimation*, contiennent des éléments et des attributs pour intégrer une animation sur un plan de montage chronologique et un mécanisme pour composer les effets de plusieurs animations.

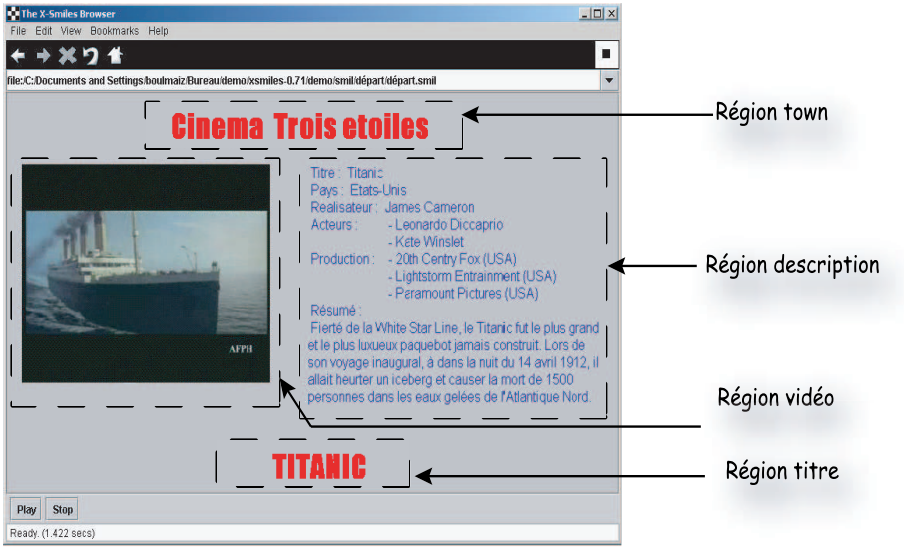

FIG. 1.4 – Exemple d'une organisation spatiale SMIL

1.3.2.4.3 Madeus

Madeus est un système d'édition et de présentation de documents multimédia [Lay02]. Son langage de présentation repose sur un modèle des intervalles de temps élastique. L'idée fondamentale consiste à offrir à l'auteur la possibilité de définir et de manipuler le document de façon à la fois explicite et flexible. Nous présentons dans l'annexe A la spécification du document SMIL présenté dans le paragraphe (§.5.2.1) en utilisant le langage Madeus.

L'intérêt de Madeus par rapport à SMIL est de deux ordres. D'abord, le positionnement temporel et spatial peut s'effectuer de façon relative. Le deuxième intérêt réside dans le fait que la spécification de la durée et de la position d'un objet se compose d'un triplet de valeurs constituant un intervalle de valeurs (borne minimum et maximum) et d'une valeur préférée [Vil02].

Dans l'exemple de la figure FIG.1.5 extrait du document présenté dans l'annexe A, on définit une durée minimale de 60s sur l'intervalle *IntervalBande*, une durée préférée de 90s et enfin, une durée maximale de 150s.

Certes, cette souplesse dans la définition des intervalles permet d'augmenter l'expressivité du langage. Cependant, Madeus n'offre aucun mécanisme pour gérer cette souplesse.

1.3.2.5 Limites

Le tour d'horizon de quelques langages de présentation effectué dans le paragraphe précédent, nous a permis d'identifier quelques lacunes de ces modèles. Un certain nombre de limites du pouvoir d'expression des langages de présentation

1. ...
2. <interval id="IntervalDesc"duration="min :60 pref :90 max :150" media="DescriptionFilm"/>
3. <interval id="IntervalBande" duration="min :60 pref :90 max :150" media="BandeAnnonce"/>
4. <interval id="IntervalMusic" duration="min :60 pref :90 max :150" media="MusicFond"/>
5. ...

FIG. 1.5 – Exemple de document Madeus

actuels sont relevées dans [BR02]. Voici quelques points qui résument ces lacunes :

- *Manque de flexibilité* : les langages de présentation permettent de spécifier des présentations relativement rigides. Les besoins actuels des présentations multimédia (cf.§.1.2.3.1) nécessitent de spécifier le placement spatio--temporel des média dans les différents axes : hauteur, profondeur, temps,... Les langages de présentation actuels offrent peu de flexibilité dans la spécification de ces axes car ces placements sont définis par valeur et non par relation. C'est le cas du SMIL qui, comme nous avons vu, ne permet que de définir des positionnements spatiaux par des valeurs absolues.
  Un premier niveau de flexibilité est déjà fourni dans quelques langages de présentation en introduisant les opérateurs d'alternatives (par exemple l'élément *switch* des langages SMIL et SVG). Cependant, le pouvoir d'expression de ces opérateurs reste limité. Dans le cas de *switch*, le choix permet de décider de la valeur d'un seul attribut (que se soit un attribut prédéfini comme : *systemScreenSize*, *systemLanguage*, *systemBitrate* ou un attribut personnalisé par l'auteur) dont la valeur est connue lors du formatage,
- *Manque du contrôle quand on a de la flexibilité* : bien qu'un certain nombre de langages de présentation offrent une certaine souplesse de présentation, ils ne donnent pas le moyen de contrôler cette souplesse. Pour expliciter ce point, prenant l'exemple de Madeus. Comme nous le mentionnons au paragraphe (§.1.3.2.4.3), Madeus permet une certaine souplesse dans la définition de la durée des éléments en permettant la spécification de la durée sous forme d'un intervalle. Mais, mis à part la possibilité de définir une valeur préférée, il n'est pas possible d'exprimer une autre stratégie s'il y a un problème de formatage. Par exemple, au lieu de diminuer la durée d'un élément, garder sa valeur préférée et supprimer un autre élément.
  Naturellement, le formateur peut tirer parti de ces intervalles de définition mais cette souplesse est la plupart du temps trop vaste pour qu'il sache la gérer sans autres informations,
- *Manque de propriétés globales de présentation* : une autre rigidité des langages de présentation actuellement est qu'ils ne permettent pas aux auteurs de spécifier des critères de présentation globaux, comme la durée

totale de la présentation, le nombre de vidéo dans une présentation. Ces critères sont très importants dans les systèmes de génération automatique des présentations multimédia. Par exemple, les auteurs dans [LGH02] proposent un système permettant la présentation des résultats d'une requête de recherche d'informations multimédia sous la forme d'un document SMIL. Il serait intéressant de donner à l'auteur les moyens pour spécifier des critères globaux de présentation des résultats.

1.3.2.6 Cas d'échec

Les limites identifiées dans le paragraphe précédent, peuvent conduire à des cas d'échec dans le processus de formatage. Nous distinguons deux cas :

- Soit le formateur est incapable de produire un résultat de formatage. Ce cas d'échec est lié aux incohérences entre les éléments et les relations (temporelles et spatiales) du document,
- Soit le résultat de formatage ne satisfait pas l'auteur. Ce cas d'échec est dû non seulement au manque de maîtrise du langage de présentation, mais aussi à la rigidité du langage de présentation qui, comme nous avons vu dans le paragraphe précédent, ne permet pas d'exprimer tous les besoins de présentation des documents multimédia.

1.3.3 Synthèse sur les langages de présentation

Les langages de présentation des documents multimédia étudiés n'ont pas les mêmes possibilités de description et n'offrent pas le même niveau de souplesse. Cependant, l'expressivité de ces langages reste insuffisante. Or le formateur peut entrer en situation d'échec.

L'intérêt d'avoir des langages de présentation limités en terme de pouvoir d'expression est de simplifier le processus de formatage.

L'expérience montre que l'augmentation de l'expressivité de ces langages entraîne des problèmes très complexes à résoudre et un travail colossal au niveau des formateurs associés pour s'assurer de gérer toute la liberté offerte.

En guise d'exemple, considérons une autre fois le cas du langage Madeus qui, comme il est présenté dans le paragraphe (§.1.3.2.4.3), offert certaine souplesse dans la définition des intervalles temporels. Rien que cette souplesse entraîne des difficultés notables inhérentes à la synchronisation temporelle des éléments.

Pour faire face à ces entraves, Madeus utilise un résolveur de contraintes. Ces derniers sont complexes à mettre en œuvre et consomment des ressources non négligeables en terme de temps d'exécution et de place mémoire.

Le manque d'expressivité des langages de présentation et la complexité croissante des formateurs inhérente aux tentations d'extension du pouvoir d'expressivité de ces langages ont motivé certains travaux de recherche – tels ceux présentés dans [BR02] – qui visent à permettre à l'auteur de contrôler plus finement le comportement du formateur, en définissant des opérateurs de formatage de haut niveau en complément des langages de présentation existants. Ces éléments sont utilisés pour éviter les cas d'échec. Ce langage, qui constitue notre contexte de travail, est présenté dans le chapitre 4.

1.4 Synthèse

Dans ce chapitre nous avons fait un tour d'horizon des travaux relatifs aux documents multimédia. Notamment, la modélisation et le formatage.

Au terme de cette étude, nous pouvons avancer quelques conclusions :

- L'étude non exhaustive de quelques langages de présentation a permis de montrer les limites des langages de présentation actuels par rapport aux attentes des auteurs et des utilisateurs,
- Pour le moment, la démarche suivie pour combler les lacunes que présentent les langages de présentation est de spécifier de nouveaux langages,
- De nouvelles techniques ont toutefois permis, en augmentant l'expressivité offerte à l'auteur, des améliorations notables de la qualité du formatage. Cependant, cette augmentation de l'expressivité des langages entraîne une complexité grandissante dans la réalisation des formateurs associés.

Il pourrait être intéressant d'essayer de combiner ces langages afin de répondre aux besoins de présentation en terme de la variété et de la qualité des résultats de formatage.

Ce couplage peut être envisagé à deux niveaux : couplage au niveau des langages et couplages aux niveaux des architectures (formateurs).

Dans le chapitre suivant nous proposons un état de l'art des solutions existantes pour coupler deux langages basés sur XML.

Chapitre 2

Couplage des langages basés sur XML

2.1 Introduction

Dans le chapitre précédent, nous avons présenté le standard XML comme un moyen de décrire les modèles des documents multimédia. En particulier, le point est mis sur le processus de formatage de ces documents. Nous avons montré les limites des langages de présentation actuels et le besoin de nouvelles techniques.

Nous allons maintenant faire une brève étude de quelques méthodes de couplage des langages. Plus exactement, nous présentons dans ce chapitre quelques techniques de couplage des langages basés sur XML.

Rappelons que l'un des objectifs de ce stage est de savoir comment coupler deux langages de présentation basés sur XML. Il est donc nécessaire d'étudier ce qui est fait dans ce domaine pour connaître les forces et les lacunes de chaque technique.

Nous avons identifié au moins trois techniques pour faire le couplage au niveau des langages. Ces techniques se distinguent par les mécanisme mis en jeu et les possibilités offertes de couplage.

2.2 Définition

Nous entendons par le couplage des langages *la mise en relation des éléments définis dans différents vocabulaires XML.*

Nous allons étudier dans les sections qui suivent, trois techniques de couplages des langages basés sur XML plus en détail en nous intérressant aux possibilités qu'ils offrent pour *l'ajout*, *la suppression* et *la modification* des éléments et de leurs attributs dans un document XML. En effet, commme nous le représentons au chapitre 4, nous serons toujours, dans le cadre du langage XEF, amenés à modifier, à supprimer voir à ajouter des éléments dans le document source.

2.3 Les espaces de nommage

2.3.1 Motivations

L'utilisation des espaces de nommage est motivée par le besoin d'avoir un mécanisme de modularité qui permet d'envisager des applications du langage XML où un seul document[1] XML peut contenir des éléments et des attributs qui sont définis dans différents langages basés sur XML et qui sont manipulés par des modules logiciels indépendants. En effet, si un vocabulaire de balisage existe et pour lequel des logiciels pratiques sont disponibles, il est préférable de réemployer ce langage plutôt que de le réinventer.

L'objectif des espaces de nommage, tel qu'il est annoncé par le consortium W3C [BHL99], est de donner aux éléments et aux attributs des noms uniques, sur Internet. Cette identification permet d'avoir une connaissance mutuelle d'objets, entre plusieurs DTD, sans pour autant qu'il y ait contradiction entre l'interprétation des éléments et, surtout, leurs contenus. Par exemple, il devient possible, avec les espaces de nommage, de faire coopérer, dans un même document, des éléments SMIL et SVG : les deux spécifications utilisent un élément *text* qui sera, par exemple, codé *smil :text* et *svg :texte*. Le fait de préfixer l'élément *text* permet d'enlever toute ambiguïté.

2.3.2 Définition

Un espace de nommage est une collection de noms, identifiée par une référence d'URI (Unique Resource Identifier) [IETF98], qui sont utilisés dans les documents XML comme types d'élément et noms d'attribut[2] [BHL99].

Les espaces de nommage sont utilisés dans la syntaxe XML afin de fournir un contexte aux éléments. Ils permettent ainsi d'attacher une sémantique particulière à ces éléments en fonction de leur provenance. Les espaces de nommage doivent également être représentés d'une manière unique afin de garantir que chaque élément puisse être identifié sans la moindre ambiguïté. A cette fin, chaque espace de nommage doit se conformer à la syntaxe des URI. Ces derniers, généralement utilisés pour définir des ressources physiques (par exemple un site Web) sont, dans ce contexte, utilisées pour définir des ressources abstraites représentant le contexte dans lequel des éléments d'un document XML spécifique ont été définis. Il ne faut donc pas confondre l'URI définissant un espace de nommage avec une ressource distribuée accessible via le Web. En effet, la recommandation n'oblige pas à lier des mots d'un espace de nommage à un modèle, qu'il soit sous forme de DTD ou de Schéma.

A l'image du DNS (Domain Name System) [Moc87], l'unicité des éléments est garantie par la concaténation de l'espace de nommage de cet élément avec le nom de ce même élément. XML est donc nanti d'un mécanisme permettant de préfixer le nom des éléments avec un URI.

[1] On parle alors du document composite.
[2] Nous devons cette traduction à J.J.Solari [Sol99].

2.3.3 Principe

On peut définir un espace de nommage sur n'importe quel élément d'un document : son champ d'utilisation est alors celui du sous-arbre délimité par l'élément. Pour ce faire, il suffit d'ajouter à l'élément, un attribut spécial identifiant l'espace : *xmlns : URI_espace_de_nommage*. Le suffixe *URI_espace_de_nommage* servant de préfixe aux noms d'éléments et d'attributs appartenant à l'espace de nommage défini. Par souci de clarté, les concepteurs de XML ont choisi d'associer dans le corps du document l'URI à un identifiant. Celui-ci est souvent plus intuitif et moins long.

Un exemple concret est sans doute utile pour illustrer ce concept. Dans l'exemple présenté dans la figure FIG.2.1, nous avons défini plusieurs espaces de nommage. Entre autres, l'espace dénommé *http ://www.w3.org/2001/SMIL20/Language*, qui définit un espace de nommage pour le langage SMIL. Cette déclaration définit aussi que tous les éléments de cet espace de nommage seront préfixés par *smil*. Alors que l'espace de nommage *http ://purl.org/metadata/dublin_core#* permet de référencer les éléments du Dublin Core [DC97] et seront préfixés par *dc*.

---

1. <smil xmlns :smil = http ://www.w3.org/2001/SMIL20/Language>
2. <smil :head>
3. <smil :metadata id="meta-rdf">
4. <rdf :RDF xmlns :rdf = "http ://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns :rdfs = "http ://www.w3.org/TR/1999/PR-rdf-schema-1999-0303#" xmlns :dc = "http ://purl.org/metadata/dublin_core#" xmlns :-smilmetadata = "http ://www.example.org/AudioVideo/.../smil-ns#" >
5. <! − − Métadonnées à propos de la présentation SMIL −>
6. <rdf :Description about="http ://www.example.com/meta.smi" dc :-Title="Introduction au Cadre de Description de Ressources (Resource Description Framework (RDF))" dc :Description="RDF permet l'encodage, l'échange et la réutilisation de métadonnées structurées">
7. . . .. . .
8. <dc :Creator>
9. <rdf :Seq ID="CreatorsAlphabeticalBySurname">
10. <rdf :li>Cécile Roisin</rdf :li>
11. ...
12. < /dc :Creator>
13. <smilmetadata :ListOfVideoUsed>
14. ...
15. < /smilmetadata :ListOfVideoUsed>
16. <smilmetadata :Access LevelAccessibilityGuidelines="AAA"/>
17. < /rdf :Description>
18. < /rdf :RDF>

19. < /smil :metadata>
20. ...
21. < smil :layout>
22. < smil :region id="text" title="texte" left="0" top="0" height="50" width="320">
23. ...
24. < smil :body>
25. < smil :par title="multistream">
26. ...
27. </smil :par>
28. </smil :body>
29. </smil :smil>

FIG. 2.1 – Exemple d'un document XML contenant des espaces de nommage

Il est possible de déclarer un espace de nommage par défaut. Pour le faire, il suffit de déclarer dans un élément un attribut *namespace* sans préfixe. Dans ce cas, l'espace de nommage par défaut est sensé s'appliquer à l'élément sur lequel il est déclaré et à tous les éléments fils sans préfixe.

2.3.4 Traitement des espaces de nommage par le modèle objet de document

Comme le suggère la recommandation DOM[3] (cf.§.2.4.2), au niveau de ce dernier, l'attribut spécial *xmlns*[4] est manipulé de la même manière que n'importe quel autre attribut. Cependant, les nœuds restent attachés en permanence aux URI des espaces de nommage lors de leur création. En conséquence, le déplacement d'un nœud dans un document, en utilisant DOM, ne produit en aucun cas un changement de son préfixe d'espace de nommage ou de son URI d'espace de nommage. De la même façon, la création d'un nœud, avec un préfixe d'espace de nommage et d'une URI d'espace de nommage ou la modification du préfixe d'espace de nommage d'un nœud, ne résulte dans des quelconques ajout, retrait ou changement des attributs spéciaux pour déclarer les espaces de nommage.

2.3.5 Synthèse

Il ressort de ce qui précède que les espaces de nommage sont une facilité syntaxique pour désambiguïser des noms des éléments et des attributs, mais ne permettent pas de résoudre à eux seuls le problème d'intégrer au sein d'un seul

[3]Il s'agit des recommandations à partir de DOM niveau 2. Le DOM niveau 1 ne supporte pas les espaces de nommage.

[4]Dans le DOM, les attributs de déclaration d'espace de nommage sont par définition rattachés à l'URI de l'espace de nommage " http ://www.w3c.org/2000/xmlns/ ".

document XML des éléments et des attributs provenant de différents langages basés sur XML. Ils ne donnent pas en eux-mêmes une solution efficace et automatique pour importer des déclarations contenues dans des entités externes, détecter tout conflit de dénomination qui pourrait survenir et créer des noms préfixés non ambigus. Tout ceci est du ressort de l'application.

2.4 Techniques de transformation

La transformation est le processus qui permet de convertir un document source vers un document cible suivant des règles spécifiées dans une feuille de transformation.

Bien qu'il soit tout à fait possible d'effectuer des transformations en utilisant un langage de programmation classique, de nombreux travaux de recherche ont permis de définir plusieurs langages dédiés à la transformation de document.

Nous ne prétendons pas faire un état de l'art dans ce domaine qui dépasse largement notre cadre de recherche et le lecteur intéressé peut trouver dans la thèse de Stéphane Bonhomme [Bon98] une présentation complète. La pertinance de cet état de l'art la classification de ces langages selon leur méthode de transformation et de génération en deux catégories [Vil02, Meg01] :

- *Transformation dirigée par la source* : la transformation est conduite par un parcours en profondeur d'abord du document. Un ensemble de règles est associé à chaque événement du parcours (début d'un élément, fin d'un élément, début du document,...). Greger Lindén [Lin97], dans sa classification des langages de transformation, parle alors de *la transformation basée sur l'évènement.*
- *Transformation par requêtes* : le document cible est construit par extraction d'informations du documents source.

Nous résumons dans le tableau TAB.2.1, les différences entre ces deux approches. Comme il est montré dans ce tableau, plusieurs techniques de transformations ont été proposées. Nous présentons, par la suite, trois d'entre elles : SAX, DOM et XSLT.

2.4.1 SAX

2.4.1.1 Principe

Le principe de SAX est de parcourir le document XML, et en fonction des éléments qui le constitue, de réagir à des évènements comme le début du document XML, la fin du document, le début d'une balise, la fin d'une balise, le contenu d'une balise, pour y associer un traitement. La norme [Meg01] définit un ensemble d'interfaces permettant de traiter ces évènements.

Une application utilisant SAX implémente des gestionnaires d'évènements, lui permettant d'effectuer des opérations selon le type d'évènement généré : début de document, début d'élément, fin d'élément, ...

Catégorie	Transformation dirigée par la source	Transformation par requête
Exemple	– SAX	– DOM – XSLT
Avantages	– Rapide – Consomme peu de mémoire	– Utile pour les applications modifiant l'arbre du document.
Inconvénients	– Ne permet pas de manipuler l'arbre du document	– Lent – Consomme beaucoup de mémoire – Norme incomplète.

TAB. 2.1 – Techniques de transformation des documents XML

2.4.1.2 Synthèse

L'avantage de SAX (et généralement des transformations dirigées par la source) est la performance d'exécution. En effet :

- SAX est bien adapté à des fonctionnalités de sélection d'informations précises dans un document XML,
- SAX est capable de traiter des fichiers XML de très grande taille, puisqu'il n'opère pas sur une représentation en mémoire du document XML, mais applique des traitements au fil de la lecture du document.

Cependant, cette technique ne permet pas de transformer l'arbre lors de l'analyse (elle ne fait que consommer les données). En particulier, il est impossible de réordonner les éléments dans la structure [Vil02]. On aura donc intérêt à utiliser d'autres techniques chaque fois que le traitement à réaliser nécessite des transformations ou un accès aléatoire (ou en contexte) aux constituants du document XML. La technique du modèle objet de document, répond à ces besoins.

2.4.2 Le modèle objet de document

Dans ce paragraphe, nous décrivons le modèle objet de document DOM (Document Object Model) [LLWNRCB00], une hiérarchie d'interfaces normalisées proposée par le consortium W3C permettant aux logiciels applicatifs d'accéder à la structure des documents XML et d'en manipuler le contenu. Après une description théorique du DOM, nous discutons ses limites.

2.4.2.1 Qu'est-ce que le DOM ?

DOM est un langage normalisé d'interface (Application Programming Interface, API), indépendant de toute plate-forme et de tout langage, permettant à une application de parcourir la structure du document et d'agir dynamiquement sur celui-ci. Ainsi JavaScript et ECMAScript utilisent DOM pour naviguer au sein du document HTML, ce qui leur permet par exemple de pouvoir récupérer le contenu d'un formulaire, le modifier, ...

2.4.2.2 Interfaces de DOM

Comme son nom le suggère, DOM est un modèle objet qui représente un document XML comme une collection d'objets implémentant des interfaces.

Comme dans les langages orientés objet, ces interfaces spécifient les attributs et les comportements des objets qui les implémentent. Les interfaces DOM sont organisées en une hiérarchie d'héritage dont la racine est l'interface *Node*. Comme on peut le constater en figure FIG.2.2, les interfaces DOM qui descendent de l'interface *Node* reprennent tous les constituants habituels d'un document XML. On notera que le DOM permet de représenter tout ce que l'on peut trouver dans un document XML, c'est-à-dire aussi bien ce qui relève de la structure logique (les éléments) que d'autres composantes (attributs, entités, commentaires).

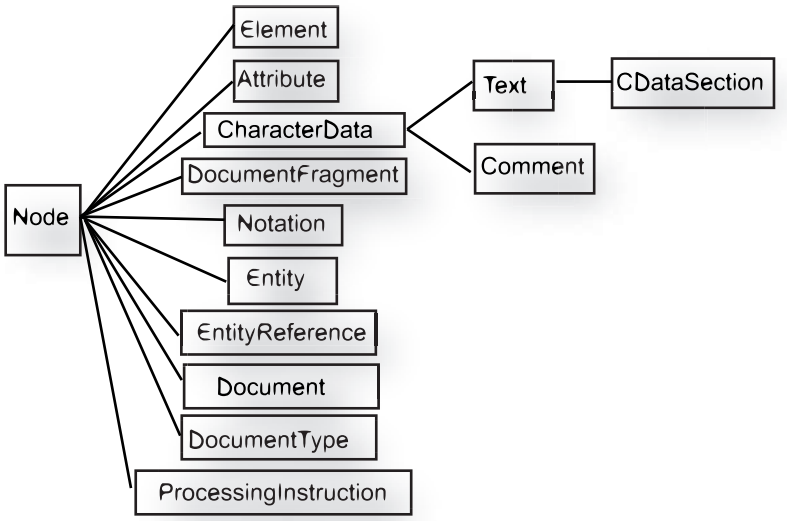

FIG. 2.2 – Hiérarchie des interfaces DOM

DOM utilise ces interfaces pour modéliser un document XML sous forme d'un arbre dont la racine est un nœud implémentant l'interface *Document*. Ce

nœud peut avoir pour fils des nœuds implémentant les interfaces *DocumentType* (si le document est accompagné de sa DTD), *Comment*, *ProcessingInstruction* et *Element*. Un nœud de type Document ne peut avoir qu'un seul fils de type *Element*. Dans l'exemple de la figure FIG.2.3, le nœud *Document* a trois descendants :

- Un nœud de type *ProcessingInstruction* qui correspond à l'instruction de traitement de la ligne 1 dans le document XML.
- Un nœud de type *Comment* représentant l'élément de la ligne 2 dans le document XML.
- Un nœud de type *Element* correspondant à l'élément *report*. Ce nœud même, est le père de deux autres nœuds de type *Element* : *title* et *description*.

```
1. <?xml version= "1" >
2.  <!– Un exemple illustrant la technique du DOM –>
3.  <report>
4.   <title> Présentation du DOM</title>
5.   <description> Cet article décrit le Document Object Model (DOM)
6.   < /description>
7.  < /report>
```

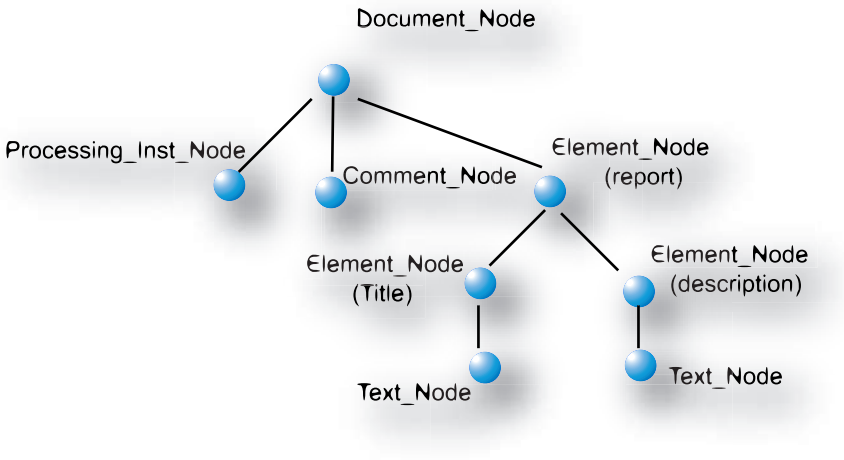

FIG. 2.3 – Exemple d'un document XML et de sa représentation DOM

2.4.2.3 Utilisation du DOM pour le couplage

DOM - tout comme SAX - permet de coupler les langages en recourant à la programmation de bas niveau. En effet, il faut écrire du code dans un langage

de programmation classique (comme Java et C) pour manipuler les documents à coupler (lecture de contenu, manipuler leurs structures voir générer un nouveau document). Cependant, Les analyseurs utilisant l'interface DOM souffrent du fait que cette API impose de construire un arbre contenant l'intégralité des éléments du document en mémoire quelque soit l'application. Ainsi pour de gros documents, DOM devient lourd à gérer. De plus, cela rend l'utilisation de DOM lente, c'est la raison pour laquelle la norme DOM est généralement peu respectée car chaque éditeur l'implémente selon ses besoins, ce qui provoque l'apparition de nombreux analyseurs utilisant des interfaces propriétaires.

Un autre reproche formulé envers cette technique, est que la norme DOM est incomplète. En effet, elle ne définit pas, entre autres, comment construire l'arbre DOM. Chaque analyseur implémente donc sa propre API. Il est par conséquent impossible d'écrire du code indépendant du parseur utilisé.

On aura par contre intérêt à utiliser DOM chaque fois que le traitement à réaliser nécessite un accès aléatoire ou en contexte aux constituants du document. DOM est adapté à la programmation d'applications interactives (par exemple : éditeurs graphique des documents XML) dans lesquelles la totalité de l'arbre du document est susceptible d'être modifié.

Signalons enfin que DOM recouvre fonctionnellement certaines autres normes de XML, même si ces dernières ont une vocation très différente. C'est par exemple le cas entre DOM et XSL[5] [XSL1], qui permettent tous les deux d'effectuer des transformations d'arbres complexes. Dans le cas de XSL, la transformation s'effectue via un langage déclaratif, alors que DOM relève d'une approche procédurale. XSL se base sur un langage de transformation appelé XSLT que nous décrivons dans le paragraphe suivant.

2.4.3 Langages de transformation

Dans l'état actuel de la recherche, un certain nombre de langages de transformation ont été proposés comme : XSLT[Cla99], Circus [Dur99] et XDuce [HP02]. Par la suite, nous illustrons le principe des langages de transformation à travers le langage XSLT.

XSLT (eXtensible Style Language Transformation) est un standard du W3C permettant de transformer un fichier XML en un autre fichier, XML ou d'un autre format (HTML, PDF, Latex ou texte par exemple). Un fichier XSLT est lui-même un fichier XML opérant une transformation sur l'arbre d'un document XML.

XSLT n'est pas un langage autonome, il est conçu comme une partie de XSL, le langage des feuilles de styles de XML. En plus de XSLT, XSL inclus un vocabulaire XML pour le formatage, XSL-FO.

XSL spécifie les règles de présentation d'un document XML en utilisant XSLT pour décrire comment le document peut être transformé en un autre document qui utilise XSL-FO. Cependant, XSLT peut être utilisé indépendamment.

La différence principale entre les transformations réalisées avec XSLT par rapport à d'autres transformations, telles que celles intégrées immédiatement

[5]eXtensible Stylesheet Language.

dans un code exécutable (en utilisant DOM par exemple), est qu'il suffit de décrire le résultat à obtenir dans une feuille de style, puis fournir la description à un processeur[6] XSLT, plutôt que de devoir spécifier comment obtenir le résultat. C'est donc le processeur XSLT qui se charge d'effectuer les transformations proprement dites.

2.4.3.1 Principe

Le processeur XSLT crée une structure logique arborescente (on parle d'arbre source) à partir du document XML et lui fait subir des transformations selon *les règles modèles*[7] contenues dans la feuille XSLT pour produire un arbre résultat représentant, par exemple, la structure d'un document XHTML. L'arbre source peut être entièrement remodelé et filtré ainsi qu'enrichi d'un contenu qui sera propre à l'arbre résultat, si bien que l'arbre résultat peut être radicalement différent de l'arbre source.

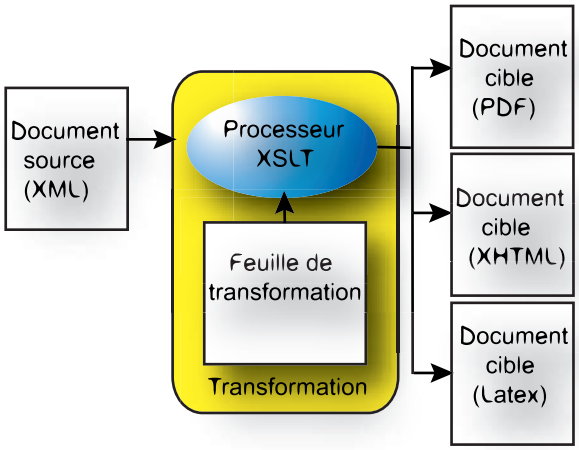

FIG. 2.4 – Principe d'une transformation XSLT

Ce processeur XSLT lit la feuille de transformation puis le document source XML et enfin applique les règles de transformations contenues dans la feuille de transformation afin de produire le document cible (cf. FIG.2.4).

Le concept de *template* est une notion clé de XSLT. Chaque *template rule* définit des traitements à effectuer sur un élément (noeud ou feuille) de l'arbre source, dans la terminologie XSLT, ces éléments sont appelés *patterns*[8]. Ces derniers, sont spécifiés à l'aide de l'attribut *match* associé à l'élément *template*.

[6] Le composant logiciel chargé de la transformation.
[7] En anglais, template rules.
[8] En français, le terme *motif* est utilisé.

Dans l'exemple ci-dessous, nous donnons une feuille de transformation permettant de générer un document XHTML à partir d'un document XML. Celui-ci, décrit les programmes des cinémas d'une ville. Nous illustrons le processus de transformation plus en détail dans le chapitre 5.

```
1. < ?xml version= “1.0” encoding= “ISO-8859-1” ?>
2.  <cinemas>
3.   <cinema>
4.   ...
5.   < /cinema>
6.   <cinema>
7.    <name>Trois étoiles</name>
8.    <adress>4 Boulevard LaChapelle </adress>
9.    <program>
10.    <film>
11.     <title>La guerre des géants</title>
12.     <kind> suspense</kind>
13.     <producer>   <producer>
14.    < /film>
15.   < /program>
16.  < /cinema>
17.  <cinema>
18.  ...
19.  < /cinema>
20.  < /cinemas>
21. < /xml>
```

FIG. 2.5 – Le document XML *Programme des cinémas*

```
1. < ?xml version= “1.0” encoding= “ISO-8859-1” ?>
2.  <xsl : stylesheet xmlns : “http ://www.w3c.org/1999/xsl/Transform”>
3.   <xsl : attribute-set name = “MonStyle”>
4.    <xsl :attribute name= “font-name”>helvetica</xsl :attribute>
5.    <xsl :attribute name= “font-size”>14pt</xsl :attribute>
6.    <xsl :attribute name= “font-style”>italic</xsl :attribute>
7.   < /xsl :attribute-set>
8.   <xsl :template match= />
9.    <html>
10.    <head>
```

```
11.      <title>Cinema program in Paris</title>
12.     < /head>
13.     <body xsl :use-attribute-sets= “MonStyle”>
14.      <xsl :apply-template select=”cinema”>
15.      < /body>
16.    < /html>
17.   < /xsl :template>
18.   <xsl :template match= “cinema”>
19.    Le programme du cinéma : <xsl :value-of select=”name”/> est :<br>
20.    <b>Film : <xsl :value-of select= “title”/>   </b></br>
21.    Le genre du film, c’est : <i><xsl :value-of select= “kind”/></i><br>
22.   < /xsl :template>
23.  < /xsl :stylesheet>
```

FIG. 2.6 – Feuille de transformation du document XML *Programme des cinémas*

```
1. <html>
2.  <head>
3.   <title>Cinema program in Paris </title>
4.  < /head>
5.  <body font-name=  “Helvetiva” font-size= “14pt” font-style= “italic”>
6.   Le programme du cinéma <b>Trois étoiles</b> est : <br>
7.   <b>Film : La guerre des géants </b><br>
8.   Le genre du film, c’est :
9.   <i>suspense</i><br>
10.  < /body>
11. < /html>
```

FIG. 2.7 – Le document HTML *Programme des cinémas*

2.4.3.2 Utilisation de XSLT pour le couplage

Une limite majeure de XSLT, et d’ailleurs de tous les langages de transformation actuels, est qu’il ne garantit pas la validité de document produit par rapport à un modèle de document.

En outre, on peut reprocher à XSLT, l’impossibilité d’ajouter un attribut à un élément après que des descendants lui aient été rajoutés.

2.5 Langages de sélection

2.5.1 Principe

Bien que DOM fournisse un moyen précis et puissant d'effectuer les traitements de navigation et de transformation sur un document XML, il implique un recours systématique à la programmation qui s'avère assez lourd. D'autres techniques, plus abstraites mais limitées au niveau des traitements qu'ils permettent, ont été proposées pour naviguer dans les documents XML. Il s'agit des langages de sélection.

Plusieurs langages de sélection ont été proposés, comme le langage de sélection [GCLW01] de CSS ou celui de XPath, pour ne citer que quelques exemples. Le langage de sélection de CSS s'appuie uniquement sur le type du nœud source et sur son contexte hiérarchique. De plus, le langage de sélection est utilisé uniquement pour mettre en correspondance un nœud source et une règle de transformation [Vil02].

Le consortium W3C a défini un nouveau langage de sélection. Le langage XPath [CD99] fournit des expressions concises pour designer des nœuds dans un arbre DOM à l'aide de chemins. Par rapport au langage de sélection de CSS, XPath est plus expressif puisqu'il permet notamment d'exprimer des sélections en fonction du contexte d'exécution au moment de son évaluation.

XPath sert en fait à d'autres langages de balisage basés sur XML :

- *XSLT* : XSLT utilse des expressions XPath pour sélectionner les éléments du document source à transformer (cf.§.2.4.3).
- *XPointer* [GMMW03] : un langage utilisé pour référencer des fragments d'un document XML. XPointer s'appuie sur le langage XPath pour permettre une identification précise dans la structure des documents XML.
- *XQuery* [BCFFRS02] : Langage de requête permettant d'accéder à chacun des éléments d'information d'un document XML, d'en sélectionner des listes et de les manipuler.
- *XForms* [DKMR02] : XML Forms est une spécification de définition de formulaires, basée sur XML. XForms utilise XPath pour adresser les instances des nœuds de données, exprimer les contraintes et enfin pour spécifier les calculs.

XPath s'appuie sur une modélisation DOM restreinte, seuls certains types d'éléments sont considérés. Nous avons présenté dans la figure FIG.2.8, le sous-ensemble des interfaces DOM qui n'est pas pris en compte par XPath par un fond gris. XPath ne prend pas en compte des nœuds de type *DocumentType*, on ne peut donc pas traiter avec XPath la DTD d'un document XML. De même pour les références vers des entités externes (les nœuds de type *Entity* et *EntityReference*). Concrètement, cela signifie qu'une application faisant appel à des expressions XPath (un processeur XSLT par exemple) doit résoudre au préalable les références aux entités et ne pas prendre en compte la DTD.

En ce qui concerne les sections latérales (nœuds de type *CDataSection*), les simplifications suivantes sont effectuées :

- Le contenu des nœuds *CDataSection* est transformé en texte, tout se passe comme si ce type de nœud était en fait de type Text, avec un contenu

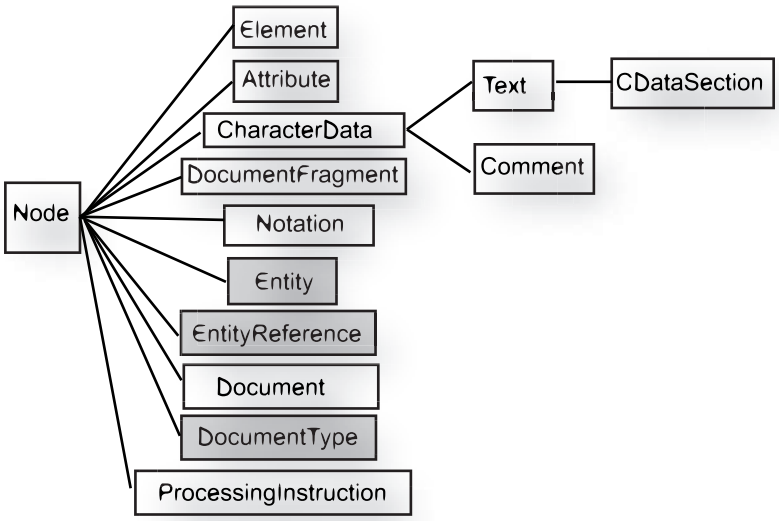

FIG. 2.8 – Le sous-ensemble des interfaces DOM pris en compte par XPath

transmis tel quel par le parseur.
- Si plusieurs nœuds texte sont frères et sont adjacents dans l'arbre DOM, ils sont réunis dans l'arbre XPath.

Les chemins de localisation

Le langage XPath désigne un ou plusieurs nœuds en exprimant des chemins de localisation. L'évaluation d'un chemin donne un résultat qui peut être soit une valeur numérique ou alphanumérique, soit un sous-ensemble des nœuds de l'arbre. Cette évaluation tient compte d'un contexte qui détermine le résultat.

Un chemin peut être absolu et prendre son origine à la racine du document. Il peut également être relatif et prendre pour origine un nœud du document, dit nœud *contexte*.

Un chemin est constitué d'une suite d'étapes, séparées par des "/". Le format général d'un chemin est donc : chemin : : *[/] étape / étape / .../ étape*

Le "/" initial est optionnel, et distingue les chemins relatifs des chemins absolus, comme nous l'avons expliqué précédemment.

Chaque étape peut elle-même se subdiviser en trois composants : Axe : : filtre [prédicat1] [prédicat2]...

- *L'axe* : définit le sens de parcours des nœuds à partir du nœud contexte. nous présentons dans la figure FIG.2.9, la sémantique des axes définis dans la recommandation XPath [CD99],

– *Le filtre* : indique le type des nœuds qui seront retenus dans l'évaluation du chemin. La notion du type recouvre ici les types DOM (texte, instruction de traitement,...) ainsi que, pour les éléments et les attributs, le nom de la balise ou de l'attribut.
– *Les prédicats* : expriment les propriétés que doivent satisfaire les nœuds retenus à l'issue du filtrage pour être finalement inclus dans le résultat.

Dans le document XML présenté dans la figure FIG.2.5, le chemin *child : : film [position ( ) = last ( )]*, Permet de sélectionner le dernier fils de type *film* du nœud courant. Les trois composants d'une étape apparaissent dans cet exemple, l'axe *child* permet de sélectionner les nœuds fils direct du nœud courant. Le filtre *film* permet d'en extraire les nœuds dont le nom est *film*. Enfin, le prédicat : *position ( ) = last ( )*, garde le dernier élément *film*.

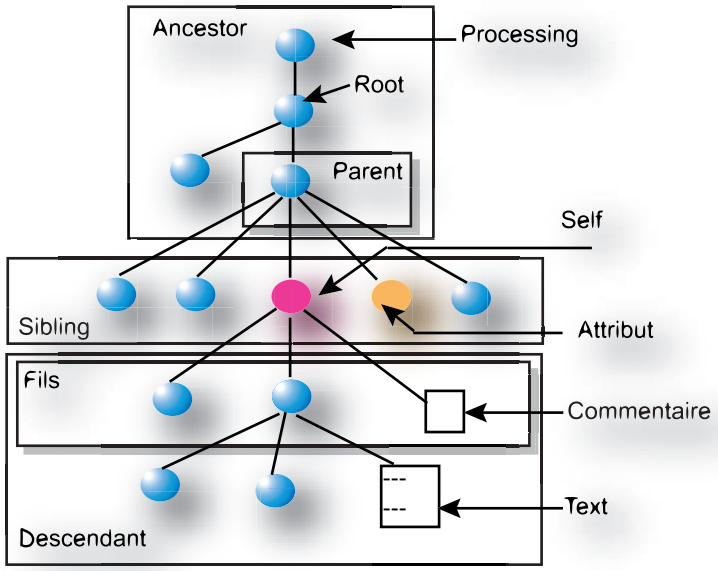

FIG. 2.9 – Exemples d'axes dans XPath

2.5.2 Utilisation des langages de sélection pour le couplage

Les langages de sélection et en particulier XPath permettent d'identifier les noeuds d'un arbre de manière concise. Cependant, ces langages ne permettent pas la modification (l'ajout, la suppression et la modificaton des valeurs des propriétés) de cet arbre.

2.6 Synthèse

Nous avons présenté dans ce chapitre trois techniques de couplage des langages basés sur XML. La première est la technique **des espaces de nommage** qui permet d'utiliser des ensembles de balises prédéfinies (langages de balisage) dans un seul document composite et d'enlever les ambiguïtés possibles lors de l'utilisation d'une balise. La seconde est **la technique de transformation** qui propose de coupler les documents sources en les convertissant vers un document cible. Enfin la troisième technique, l'utilisation des **langages de sélection** permet d'identifier les éléments du document source à manipuler.

Chaque technique a ses avantages et ses inconvénients au regard du coût, de l'efficacité et du pouvoir de traitements qu'elle permet. D'un point de vue de l'efficacité, ce sont les techniques de transformation qui l'emporte. En effet, ces techniques permettent d'effectuer des traitements complexes sur le document source. En particulier, l'ajout et la suppression des éléments. L'inconvénient de ces techniques concerne surtout le coût d'exécution relativement élevé par rapport aux autres techniques.

Néanmoins, nous pensons que ces trois techniques sont complémentaires et pourront être utilisées conjointement pour répondre au mieux à nos besoins de couplage des langages de présentation.

Chapitre 3

Couplage des architectures logicielles

3.1 Introduction

L'architecture logicielle a toujours joué un rôle névralgique dans la construction d'applications. En effet, si le choix de l'architecture est approprié, cela permet au produit de satisfaire les exigences et d'être facilement modifiable. Au contraire, une architecture mal pensée peut poser un certain nombre de problèmes notamment lorsque des modifications doivent être faites au système.

Pour tirer profit des logiciels existants et faciliter le développement de nouvelles applications, des techniques modernes ont vu le jour. On ne développe plus une application à partir de zéro, mais on construit des applications de plus en plus par assemblage de composants "briques" logiciels issus de développements précédents.

Rappelons que nous voulons créer un nouveau formateur XML par composition de formateurs existants, il nous semble donc intéressant de faire le bilan des travaux existants sur le couplage des architectures logicielles et ce que l'on peut espérer pour le couplage des formateurs XML.

Nous proposons dans ce chapitre de faire un point sur les architectures logicielles, en se focalisant sur le concept de la composition et les techniques sous-jacentes. Pour ce faire, nous définissons dans un premier temps la notion de composition, puis nous explicitons la notion de couplage, notion très liée à la composition et enfin, nous étudions la mise en œuvre de la composition dans le paradigme de la programmation orientée composant et nous dressons un bilan sur la composition des formateurs XML, en dernier lieu.

3.2 La composition

Cette section a pour vocation d'expliciter le concept de composition. Pour ce faire, nous présentons dans un premier temps les différentes sémantiques

données à ce concept d'un point de vue cognitif. Nous nous intéressons ensuite aux mécanismes de propagation des données dans une relation de composition.

3.2.1 Définition

Par objet composite, nous entendons un objet formé par l'agrégation d'un ensemble d'objets, appelés ses composants, qui décrivent chacun une partie de l'objet composite [GPV94, Nap92]. La relation qui lie un objet composite à ses composants est appelée relation de composition [HGP92].

De nombreuses recherches en sciences cognitives et en linguistique se sont intéressées à la compréhension de la nature de la relation de composition. En particulier, les travaux de [WCH87, CH88] ont établi une classification de la composition en sept catégories. Cette classification a été établie à base de quatre critères [WCH87] : la similarité, la séparabilité, la simultanité et la fonctionnalité.

Dans le domaine de l'ingénierie logicielle, la relation de composition la plus utilisée est la composition *Composant/Objet*. Ce type de composition traduit le fait que les composants exhibent certaines formes de relations structurelles ou fonctionnelles entre eux et le composite auquel ils appartiennent.

L'une des difficultés majeures que pose la composition est la gestion des échanges de données entre les composants d'un objet composite. Le paragraphe suivant donne un aperçu de ce problème.

3.2.2 Propagation des attributs dans la composition des applications

Le mécanisme de propagation des attributs[1] consiste à propager la valeur d'un attribut à travers les liens de composition. Lorsqu'on crée des objets composites, il est en effet souvent utile de pouvoir exprimer qu'un objet composite peut propager ses attributs à ses composants et inversement.

Afin de mieux comprendre le mécanisme de propagation des attributs, nous proposons d'étudier les caractéristiques suivantes :

- Nature de la propagation,
- Sens de la propagation,
- Type de la propagation.

3.2.2.1 Nature de la propagation

Dans la composition, on distingue deux types de propagation des attributs :

- La propagation de l'attribut et de sa valeur,
- La propagation de valeur de l'attribut.

Dans le premier cas, l'attribut propagé est considéré comme définissant une propriété aussi bien pour l'objet émetteur (composite ou composant) que pour l'objet récepteur (composite ou composant), même si ce dernier ne possède pas

[1] Appelée aussi partage de propriétés dans [Nap92].

l'attribut. Par contre, dans le deuxième cas, il s'agit de récupérer une valeur pour un attribut défini au niveau de l'objet émetteur.

3.2.2.2 Sens de la propagation

Le sens de la propagation des attributs dans un objet composite est soit :
- *Du composite vers le composant* : l'objet composite peut propager ses attributs aux objets composants,
- *Du composant vers le composite* : le composant peut propager ses attributs aux objets composites dont il fait partie,
- *Bidirectionnel :* l'objet composite peut propager ses attributs à ses composants et inversement.

3.2.2.3 Type de la propagation

La propagation des attributs peut être :
- *Totale* : tous les attributs sont propagés,
- *Sélective* : seuls les attributs spécifiés sont propagés. Dans ce cas, on considère qu'il existe des attributs propres à l'objet composite ou aux composants qui ne peuvent être partagés.

3.2.2.4 Problèmes de la propagation

Le mécanisme de la propagation des attributs soulève cependant certains problèmes tels que :
- La propagation des attributs peut conduire à des conflits analogues aux conflits d'héritage dans la technologie objet. Un conflit apparaît dés lors qu'il existe deux objets incomparables par la relation de composition et pouvant transmettre une valeur pour un attribut donné. Dans le paradigme de programmation orientée objet, les conflits sont ignorés ou considérés comme erreur,
- Le non-respect du principe d'indépendance des composants. Ce principe consiste à considérer que l'objet composite connaît ces composants mais les composants ne connaissent pas les objets composites dont ils font partie [MK89]. Pour spécifier dans un composant qu'il peut récupérer la valeur d'un attribut dans un objet composite, il est en effet nécessaire que ce composant connaisse les objets composites dont il fait partie.

3.3 Couplage des architectures

De nombreux travaux menés sur la composition des logiciels et notamment sur la réutilisation de code dans cette composition, ont permis des avancées importantes. Ainsi, plusieurs techniques de réutilisation ont été proposées[2] : la duplication, le préprocessing, les bibliothèques, les paquetages, ... La maturité de ces techniques est apparue avec le paradigme de la programmation

[2]Ces techniques sont bien détaillées dans [Cou96].

orientée composants. Avant de présenter quelques techniques de composition, nous présentons la notion de couplage, une notion très liée à la notion de composition.

3.3.1 Définition

Deux (plusieurs) objets sont couplés si l'un d'entre eux utilise les méthodes et/ou les variables de (des) l'autre(s) objet(s) [KKL00][3]*. Le couplage permet de mesurer le degré d'interconnexion des modules [SMC74].*

3.3.2 Niveaux de couplage

La notion du couplage occupe une place primordiale dans l'évaluation des logiciels. Les travaux effectués dans ce domaine ont permis de distinguer sept degrés[4] (niveaux) de couplage. Ce classement est établi sur base des critères de : la compréhensibilité, la maintenabilté, la modifiabilité et la réutilisabilité [PJ80, BBL76, Mey88]. Dans ce paragraphe, nous résumons les principales caractéristiques de chaque niveau [Led03, Lee99, Zel00, Ryo01, PJ80, Mye74].

3.3.2.1 Absence de couplage direct

Deux modules n'interagissent pas et ne partagent pas d'information. Toute évolution de l'un n'a pas d'impact sur l'autre. Naturellement, c'est le type de couplage idéal pour réduire l'influence de l'interaction entre les modules.

Exemple

- Deux modules très éloignés dans la structure de modules qui ne partagent ni types ni données.
- Deux formateurs XML différents qui agissent sur deux arbres DOM différents et qui ne se communiquent pas.

3.3.2.2 Couplage de données

Les deux modules n'échangent que des données simples (non structurées) via leurs interfaces. Le contenu des deux modules peut évoluer sans effet sur l'autre module tant qu'on ne modifie pas l'interface ; les modules ne partagent pas de définitions de types.

Exemple

- Une procédure en appelle une autre et passe ses paramètres entiers par valeur.
- Deux formateurs XML qui agissent sur deux arbres DOM dífférents et qui s'échangent les valeurs des attributs.

[3] Traduction personnelle.

[4] D'autres travaux affinent ce classement à plus de sept degrés. Nous ne les présentons pas ici et le lecteur peut se référer à des travaux comme [OHK93, JO95, EKS94] pour plus d'informations.

3.3.2.3 Couplage par références

Les modules échangent via leurs interfaces, des données structurées. Alors qu'ils n'agissent que sur une partie de ces données. Dans ce type de couplage, les modules sont sensibles aux évolutions des structures de données échangées (définitions de types). A l'exécution, ils peuvent conserver les références (pointeur, adresse d'un objet) et les détourner.

Exemple

- Passage par référence de paramètres.
- Une procédure A fait appel à une autre procédure B en lui passant un enregistrement en paramètre, alors que la procédure B n'utilise que quelques champs de cet enregistrement.
- Un formateur XML passe à un autre formateur la liste des nœuds fils (l'attribut *childNodes* de ce noeud) d'un nœud particulier. Alors que ce dernier formateur n'utilise que le premier fils de ce nœud.

3.3.2.4 Couplage de contrôle

L'interface du module permet d'influencer son comportement et le module appelé peut contrôler la logique du module appelant. Autrement dit, Le module appelé détermine la future séquence du module appelant à exécuter. Ici les modules donnent de l'information sur ce qu'il y a dans la boîte. Ils hypothèquent leur liberté d'évolution.

Exemple

- Un module de tri à qui on passe en paramètre la méthode de tri,
- Un formateur XML (par exemple SVG) passe un attribut de test personnalisé à une instruction *switch* (cf.§.1.3.2.4.2) pour un formateur SMIL. Dans ce cas, l'attribut passé détermine l'alternative de l'élément *switch* à exécuter.

3.3.2.5 Couplage externe

Les deux modules communiquent par l'intermédiaire d'un environnement extérieur à l'application. Le canal de communication n'étant pas identifié, il risque d'être oublié lors de l'évolution.

Exemple

Deux applications de scolarité :

- Inscriptions administratives
- Feuille de calcul des résultats

Les communications se font par l'intermédiaire d'utilisateurs (enseignants, copies d'examen).

3.3.2.6 Couplage commun

Un couplage est commun lorsque les modules communiquent via un espace partagé de données globales non-structurées. L'utilisation de variables globales réclame une grande discipline d'utilisation. Plus une variable est utilisée, plus il

est difficile de faire évoluer sa structure. Par rapport au couplage ci dessous (cf.§ 1.3.2.7), dans le couplage commun, toutes les communications entre les modules s'effectuent via l'espace partagé.

Exemples

– Communication à base de variables partagées.

3.3.2.7 Couplage de contenu

Un module connaît et exploite le contenu de l'autre module (accès direct à des variables privées, à la structure logique,...). Une fois que le contenu d'un module est connu et exploité, il ne peut plus évoluer.

Exemple

– Utilisation de la valeur de constantes.
– Le mot clé *friend* en C++ permet de déclarer une fonction ou une classe avec les droits d'accès complets aux membres privés et protégés de la classe, sans être un membre de cette classe. La classe externe dispose d'un accès complet à la classe qui la déclare *friend*.
– On peut choisir une architecture pour coupler un formateur SMIL et un formateur SVG où le formateur SVG accède au contenu du formateur SMIL pour formater un élément *animate.*

3.3.3 Synthèse

Bien que les limites entre les différents niveaux du couplage dans la classification précédente ne soient pas très claires, ils chevauchent même, elle montre l'impact de degré du couplage sur l'évolution d'applications.

Idéalement, le couplage entre les modules d'une application bien conçue doit être faible. En effet, un couplage faible entre les composants laisse plus de liberté pour modifier l'application composée.

Page-Jones [PJ80] avance le fait qu'il y a au moins 3 raisons de croire qu'un couplage faible entre les modules d'une application est préférable :

– Peu d'intercommunications entre les modules réduit le risque qu'un défaut dans un module causera un échec dans d'autres modules,
– Peu d'intercommunications entre les modules réduit le risque que des changements d'un module causeront des problèmes dans d'autres modules, ce qui augmente la réutilisabilité,
– Peu d'intercommunications entre les modules réduit le temps du programmeur pour comprendre les détails d'autres modules.

Suivant l'ordre établi précédemment, la force de couplage se change de manière croissante (cf.FIG.3.1) et donc d'un point de vue de la conception d'une architecture évolutive, ces niveaux de couplage vont du meilleur cas au pire cas (l'architecture sera difficile à faire évoluer, car les composants sont trés dépendants les uns des autres).

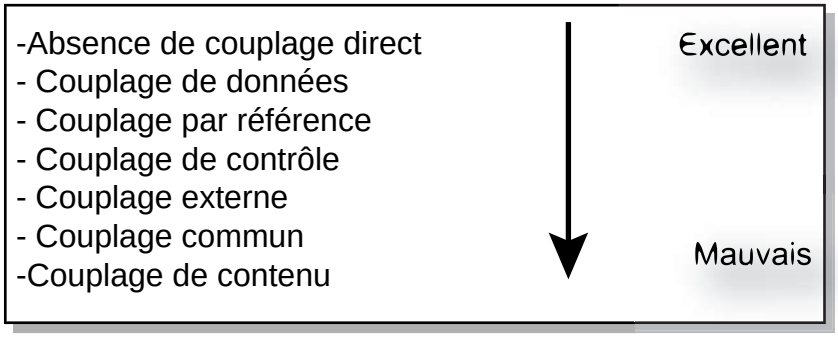

FIG. 3.1 – Niveaux de couplage

3.4 Composition des composants

Après la présentation théorique des concepts de composition et de couplage, nous allons maintenant étudier ces concepts à travers les différents paradigmes de programmation. En particulier, nous nous intéressons aux techniques de composition des composants.

Dans ce qui suit nous motivons et présentons d'abord le paradigme de la programmation orientée composant. Puis, nous présentons différentes approches permettant de mettre en œuvre la technique de la composition, dont nous dressons un bilan en dernier lieu.

3.4.1 Objets versus composants

Rappelons ici que notre but n'est pas de faire une étude comparative entre les techniques de programmation[5]. En effet, il existe dans la littérature beaucoup d'études comparant ces différentes techniques et nous recommandons à ce propos [Wat90] aux lecteurs intéressés. Il n'est cependant prouvé nulle part qu'une méthode est plus efficace qu'une autre. Par contre du point de vue de la maintenance du logiciel et surtout de la réutilisation, elles ne sont pas toutes équivalentes. A notre avis deux seulement de ces techniques prétendent résoudre le problème de réutilisabilité et le présente comme l'une de leurs caractéristiques : c'est *l'approche objet*[6] et *l'approche composant.*

Du fait de ses nombreux atouts (encapsulation, modularité,...), la programmation par objets constitue indéniablement une technologie importante pour aider à la construction d'applications complexes. Elle favorise la réutilisation de parties de logiciel déjà développées [Mey97] et permet ainsi la réduction des

[5]Une technique de programmation est un ensemble de règles plus ou moins formelles définissant l'écriture d'un programme.

[6]Nous n'entrerons pas dans le détail de ce paradigme de programmation, et le lecteur intéressé pourra se référencer à [Ous97] pour obtenir plus d'information.

délais de développement et de maintenance. Cependant, cet objectif n'est pas toujours atteint [AWY93, GHJV95, Kic96, SLMD96]. En effet :

- La technologie orientée objet s'appuie sur l'héritage d'implémentation (par lequel les nouveaux objets héritent de l'implémentation des méthodes d'un objet existant). Certes, c'est un mécanisme fondamental du code réutilisable, toutefois, nous pensons que cette forme d'héritage est insuffisante pour permettre la réutilisation dans un environnement hétérogène. En dépit de l'isolation fournie par les interfaces, les modifications des objets de base pourraient avoir des effets inattendus sur les objets qui héritent d'eux,
- Bien que la technologie orientée objet permette d'une part, d'exprimer les interfaces des entités logicielles manipulées, résultat de la conception, et d'autre part, de bien séparer la définition de ces interfaces de l'implémentation des entités logicielles, elle n'est pas adaptée à la description des schémas de coordination et de communication, elle ne spécifie pas l'interaction entre les objets,
- le code fonctionnel est souvent mélangé avec le code non fonctionnel,
- Enfin, l'une des raisons principales est que la réutilisation est envisagée trop tard dans le cycle de vie du logiciel.

Pour combler les lacunes de l'approche objet, des structures plus conséquentes doivent être candidates à la réutilisation [GHJV95]. L'ingénierie logicielle orientée composant est née de ce constat d'échec.

3.4.2 Architectures logicielles orientées composant

La phase de conception d'une application orientée composant a pour objectif de définir l'architecture logicielle du système étudié. Shaw et Garlan [SG65] décrivent une architecture logicielle de la manière suivante : " *Software architecture [is a level of design that] involves the description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints on these patterns* " .

Les éléments autour desquels les systèmes sont construits sont les *composants*. Ces derniers sont des unités compilables [MT00], cachant les détails de leur implémentation [MN98], et qui interagissent avec le monde extérieur à l'aide de leurs *interfaces*. Ils peuvent être déployés indépendamment par une tierce personne, ce qui permet un haut pouvoir de réutilisation [Szy97].

3.4.2.1 Définition d'un composant

Il n'existe pas de réel consensus sur une définition du concept de composant. Les définitions sont nombreuses et les chercheurs se rattachent souvent à celle qui s'accorde le mieux avec leurs préoccupations. Nous nous en tenons à une définition restrictive[7] : "*A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties*" [SP96].

[7]Pour une liste complète de définitions de composant, le lecetur peut consulter [BW98]

Cette définition, met l'accent sur des points essentiels : l'expressivité des interfaces et des dépendances, ainsi que la modularité, le déploiement unitaire et la composition par des tiers.

Un type de composant est caractérisé par trois éléments : ses interfaces, les modes de coopération avec les autres types de composants et ses propriétés configurables.

Par la suite, nous allons nous attacher aux caractéristiques qui permettent la composition de ces composants à savoir : la spécification des interfaces et les modes de coopération.

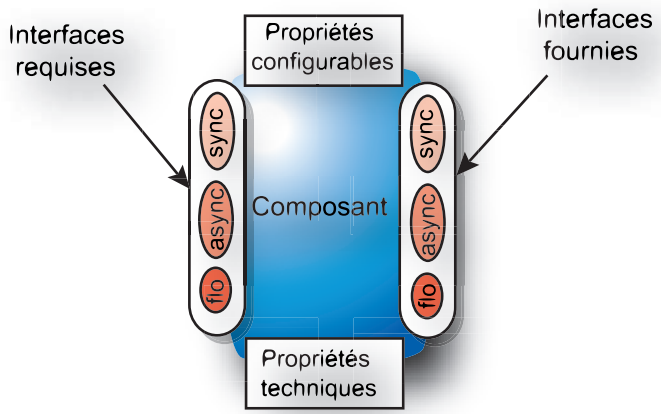

FIG. 3.2 – Le modèle de composant

3.4.2.2 La composition

L'intérêt majeur des composants réside dans le fait qu'ils sont des briques de base configurables qui vont permettre de construire des applications par *composition*. Un composant peut communiquer avec un autre composant à l'aide d'une entité appelée *connecteur* qui définit le protocole d'interaction. La connexion entre le composant et le connecteur s'effectue à travers des points névralgiques, appelés interfaces de composants.

3.4.2.2.1 Les interfaces

Les interfaces sont un élément essentiel dans le processus de composition. En effet, en plus de leur fonction de dialogue avec le monde extérieur, elles spécifient les services fournis par le composant [MT00].

Il est important qu'une interface soit définie de manière indépendante à toute implémentation, il est ainsi possible d'une part, de fournir plusieurs implémentations pour un même type de composant et d'autre part, une implémentation

de composant peut être substituée à une autre implémentation du même type.

Les interfaces d'un type de composant peuvent être de deux types :

- *Les interfaces fournies* par le composant sont du même ordre que les interfaces des objets : elles définissent les services fournis par le type de composant, en listant les signatures des opérations fournies ainsi que les différentes données entrantes et/ou sortantes du composant,
- *Les interfaces requises* par le type de composant représentent un progrès par rapport à l'approche objet. Dans le cas des objets, une référence sur l'objet utilisé est enfouie au cœur du code. Dans le cas des composants, les interfaces des types de composants utilisés sont exprimées au niveau du type de composant. Il est ainsi plus aisé d'une part, de substituer un composant par un autre et d'autre part, de gérer les connexions entre des types de composants.

Un composant coopère avec les autres composants par l'intermédiaire de services fournis et de services utilisés à travers les interfaces fournies et les interfaces requises. Pour atteindre cet objectif, il est nécessaire de définir pour chaque type des interfaces citées ci-dessus, le mode de coopération. On distingue trois modes de coopération : le mode *synchrone* (par exemple l'invocation de méthodes), le mode *asynchrone* (l'utilisation des événements) et le mode de *diffusion en continu* (les flots de données).

3.4.2.2.2 Communication inter-composants

Si on regarde au niveau conceptuel comment assembler deux composants, une première approche assez naïve consiste à relier une interface d'un composant fournissant un service à un autre composant nécessitant ce service. La représentation de cette approche est typiquement un schéma (cf.FIG.3.3) dans lequel des carrés (matérialisant les composants) sont reliés entre eux par des liens (représentants les interfaces). Cependant deux problèmes se posent.

- Le premier apparaît lors de la phase suivante du cycle de vie du logiciel. En effet, lors de l'implémentation, on peut se trouver confronté à des composants écrits dans différents langages de programmation, ou ayant des interfaces incompatibles. Or, la phase de conception décrit bien qu'il faut faire interagir deux composants mais ne dit pas comment,
- Le second problème est que lors de la phase d'analyse, des contraintes ont pu être posées et cette simple représentation ne permet pas de les garantir. Pour ces raisons, cette approche est insuffisante.

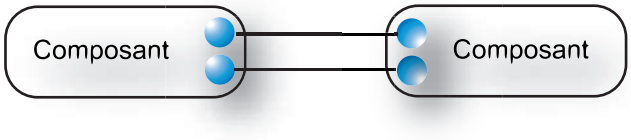

FIG. 3.3 – Liaison direct des composants

Pour pallier à ce manque, la solution proposée est d'utiliser des compo-

sants spécialisés dont le rôle est de gérer les communications inter-composants. Ce sont des *composants de communication* [Car00]. Dans ce contexte, Allen et Garlan [AG94] distinguent entre les relations d'implémentation et les relations d'interaction : "*les relations d'implémentation sont concernées par la manière de réaliser les calculs d'un composant, alors que les relations d'interaction décrivent l'interconnexion des composants dans le système*".

Plusieurs approches ont été proposées pour modéliser le concept de composants de communication :

- Un certain nombre de chercheurs défendent l'approche des *connecteurs*. Les connecteurs sont des blocs de construction architecturale utilisés pour modéliser les interactions entre les composants et les règles qui gouvernent ces interactions [MT00]. [MN98] et [SDKRYZ95] parlent de *médiateurs pour l'interconnexion*. Les divergences entre ces approches portent sur la forme que doivent prendre ces connecteurs et sur la nature de leur rôle. En effet, certains les considèrent uniquement comme des entités abstraites, n'ayant pas de forme exécutable [MT00] qui donc, n'ont de sens qu'au niveau de la conception. Au contraire, [DR97] se positionne sur le fait que si les connecteurs ne sont pas des éléments exécutables, cela peut impliquer deux choses :
  - Il peut y avoir une perte d'information entre la conception et le développement (par exemple perte d'information sur les relations entre éléments du système) car l'information spécifiée au niveau du connecteur n'est alors pas reportée.
  - Il se peut également que certaines informations vont devoir se retrouver au sein même du composant (le comportement du connecteur se retrouve au niveau du composant) ce qui implique sa modification.
- Une autre notion proposée est celle de *glu* [SN99], aussi appelée *encapsulateur* (wrapper) dans [TV01] et de script pour mettre en oeuvre la notion de connecteur lors de la phase d'implémentation. La glu a pour rôle d'enrober le composant à intégrer de manière à rendre possible la mise en relation d'interfaces incompatibles. Le langage de script a pour rôle de décrire la manière dont les composants sont reliés entre eux.

Nous donnons dans la figure FIG.3.4, un exemple de composition d'une application en utilisant des connecteurs. Dans cet exemple, le composé est une application de traitement de texte et les composants sont un *Editeur de texte* et un *Vérificateur d'orthographe*. Un connecteur (Connecteur1) a été modélisé afin de faire le lien entre les deux composants. Un autre connecteur (Connecteur2) a été utilisé pour gérer la communication entre l'application de traitement de texte et l'application tableur.

3.4.2.2.3 Techniques de composition

La composition logicielle peut être envisagée à plusieurs niveaux dans le cycle de vie de l'application. Nous nous intéressons à la phase de conception. Il s'agit de la phase durant laquelle est conçue l'architecture logicielle de l'application. Elle permet notamment de spécifier comment les composants s'intègrent dans un environnement particulier.

Actuellement, La programmation orientée composant est le seul paradigme

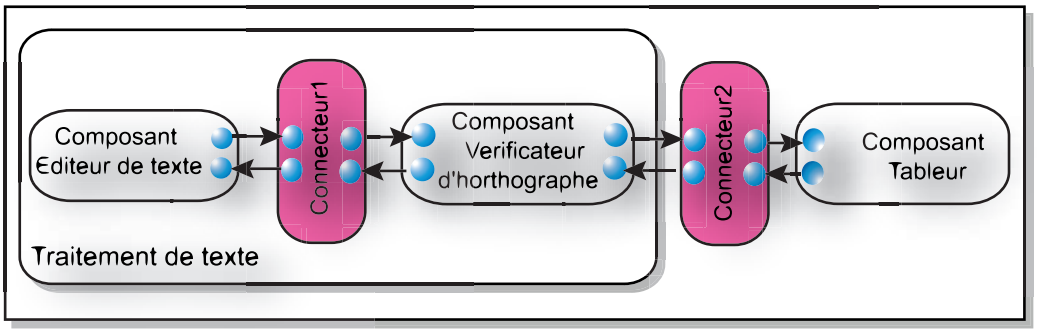

FIG. 3.4 – Exemple d'une application composée en utilisant des connecteurs

de programmation qui représente et utilise la composition de manière explicite[8].

Cette section donne un bref aperçu des travaux menés sur la composition logicielle au niveau des applications basées composants. Deux techniques de composition sont décrites par la suite : l'inclusion et l'agrégation. Ces deux techniques constituent la base de tout type de composition. Nous donnons dans le chapitre 6 quelques types de composition en cours d'élaboration.

3.4.2.2.3.1 L'inclusion

L'inclusion consiste à modéliser de nouvelles interfaces propres au composé et à cacher celles des composants.

Dans l'inclusion l'objet externe agit comme un client de l'objet interne. Comme il est présenté sur la figure FIG.3.5, l'objet externe invoque les méthodes de l'objet interne pour son propre compte, mais cela ne rend pas pour autant ces méthodes visibles à son client. Au lieu de cela, quand un client invoque une méthode dans l'une des interfaces de l'objet externe, l'exécution de cette méthode peut comprendre un appel à une méthode d'une interface de l'objet interne. En d'autres termes, l'interface de l'objet externe contient des méthodes qui appellent celles de l'objet interne.

L'objet client peut toujours invoquer les autres méthodes de l'objet interne non couvertes par l'objet externe.

Cette approche a l'avantage de rendre le composé totalement opaque en cachant totalement ses composants [BBB01].

3.4.2.2.3.2 L'agrégation

L'agrégation permet à un objet externe de présenter comme sienne une interface qui se trouve en réalité implémentée dans un objet plus interne. En effet,

[8]Récemment, le paradigme de programmation par aspects met lui aussi le point sur la composition explicite des applications.

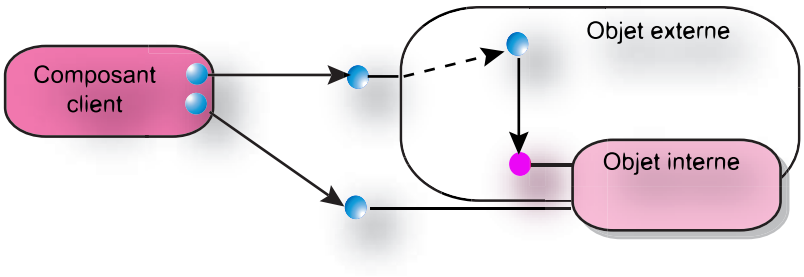

FIG. 3.5 – Composition des objets par inclusion

celle-ci permet de modéliser la relation de composition comme un objet de haut niveau mettant à disposition de clients, un ensemble d'interfaces fournissant ou nécessitant des services. Ces interfaces peuvent être propres à l'objet de haut niveau ou bien être celles des objets composants, mais sont vues de la même manière par les clients : il y a encapsulation [Lew98].

Une fois implémentée, cette technique présente l'avantage de réduire la charge de calcul grâce à l'optimisation des accès aux interfaces de ces composants.

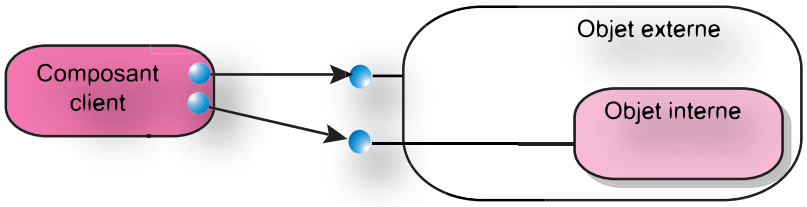

FIG. 3.6 – Composition des objets par agrégation

3.4.3 Synthèse

La programmation orientée composant raffine la technique de programmation orientée objet pour faire face aux problèmes de couplage. Une notion fondamentale des composants est les interfaces. celles-ci sont censées imopser une forte restrictions sur les interactions et la collaboration entre les composants, Par exemple en définissant le protocole d'interaction entre les composants. Cependant, aucun modèle de composant actuel n'intègre une notion d'interfaces aussi forte. En guise d'exemple, considérons le cas du modèle des Entreprise

Java Beans (EJB)[DYK01]. Dans ce modèle, les interactions entre composants dépendent en grande partie des interfaces Java *remote* et *home* des composants. Or, ces interfaces Java définissent uniquement les services qu'offrent les composants et elles ne précisent pas le protocole d'interaction entre les composants.

Récemment, quelques travaux proposent d'intégrer les protocoles d'intéraction dans les interfaces des composants. Ces travaux ont débouché sur la proposition de plusieurs modèles comme ceux présentés dans [FGS02, PV02, Wyd01]. Les techniques adoptées pour la prise en charge des protocoles d'interactions sont ad hoc, spécifiques au modèle de composants étudié.

En outre, la phase de composition comporte encore nombre de limitations ; une des causes essentielles relève des langages et des environnements de programmation actuels. Ceux-ci reposent sur l'hypothèse que les composants sont compatibles entre eux [Bos97] alors qu'un aspect séduisant de la programmation orientée composant est justement de pouvoir utiliser des *composants sur l'étagère*. Ces composants possèdent la particularité d'être développés par de tierces personnes sans concertation et ils sont donc potentiellement non compatibles. Notamment, la définition des types non compatibles.

3.5 Composition des applications centrées documents

Vu les avantages qu'elle offre, il pourrait être intéressant d'essayer d'appliquer les principes de la programmation à composants dans le domaine des documents électroniques.

L'idée vient d'une constatation toute simple : nous utilisons tous les jours de nombreuses applications pour le traitement des documents. Une grande partie de leurs fonctionnalités est redondante (correcteurs d'orthographe, dictionnaire de synonymes, calcul...). Pourquoi alors ne pas " tronçonner " ces applications en de nombreux composants (tableur, traitement de texte...), chaque document faisant alors appel aux composants dont il a besoin. Par exemple, il suffit de créer un document avec un traitement de texte, puis, si on veut rajouter des images, des équations ou un tableau, il suffit de glisser sur le document des modèles provenant des autres composants (cf. l'exemple de la figure FIG.3.4).

Dans le domaine des documents électroniques, quelques techniques ont été élaborées pour mettre en œuvre le principe de la composition des composants. Le pionnier en matière de technologie centrée documents est certainement OpenDoc [App96]. Plus récemment, des nouvelles technologies comme OLE (Object Linking and Embedding) et Bonobo (le modèle de composants pour du projet GNOME) [Bon] sont apparus.

Nous allons maintenant étudier brièvement ces modèles en nous intéressant, quand ils le permettent, aux possibilités qu'ils offrent pour la composition des applications.

3.5.1 OLE

OLE (Object Linking and Embedding) est la technologie Microsoft en matière de documents composites. En effet, Il permet d'intégrer, dans une application, des objets issus d'autres applications Windows. Il peut être vu comme un ensemble de librairies et d'application pour la composition, le stockage, l'échange et l'intégration de documents Microsoft. OLE se base sur le modèle à composant COM (Component Object Model).

OLE étend les capacités de communication inter-applications offertes par la technologie DDE[9], en introduisant une approche objet.

Dans sa première version, la communication entre le document composite et ces composants se restreint à la réservation d'une partie de la zone d'affichage. En effet, quand l'utilisateur désire modifier le document composé (par exemple, un graphique *Excel*) à l'intérieur d'un document client (par exemple, un document *Word*), l'application serveur *Excel* est alors chargée et affiche le document. Une fois les modifications effectuées, l'utilisateur quitte l'application serveur *Excel* pour retourner à l'application cliente *Word* (cf.FIG.3.7). S'il fonctionne très bien, OLE 1.0 ne fournit pas réellement un mécanisme de document composite dans la mesure où il faut quitter *Word* pour aller sur *Excel*.

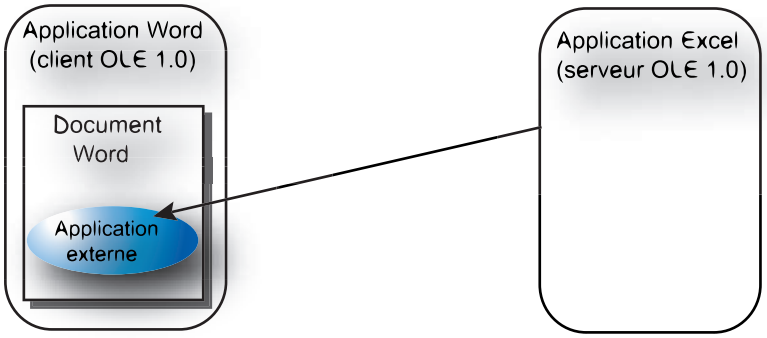

FIG. 3.7 – Exemple d'une architecture OLE 1.0

OLE 2.0 a apporté une avancée dans la mesure où cette fois, le document imbriqué est directement modifié à l'intérieur du document composite. Quand l'utilisateur double-clique sur le document imbriqué, il reste dans une fenêtre client *Word*, on parle alors d'*activation sur place*. Seuls les menus changent pour afficher les menus de l'application serveur *Excel* ; le document *Word* est toujours visible autour du graphique *Excel*. L'utilisateur n'a pas l'impression de quitter *Word*, même si *Excel* est chargé et s'occupe de la gestion graphique et des menus déroulants.

[9]La technologie Dynamic Data Exchange est la technologie Microsoft mise en œuvre dans les premières versions de Windows pour l'échange de données entre deux applications Windows.

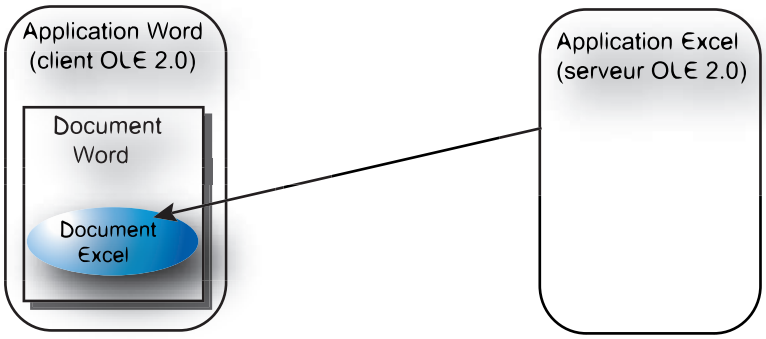

FIG. 3.8 – Exemple d'une architecture OLE 2.0

Le modèle OLE utilise le mécanisme de l'inclusion pour mettre en œuvre la composition. En effet, c'est le client OLE (l'application Word) qui doit invoquer les iméthodes du serveur OLE, donc ces méthodes sont cachées pour l'application.

Dans une composition OLE la communication entre les deux composants (le client et le serveur) se décline à des données simples (coordonnées de l'espace d'affichage). OLE propose donc un couplage faible entre les composants.

3.5.2 Le projet Bonobo

Bonobo est le modèle à composants proposé dans le cadre du projet Gnome (GNU Network Object Model Environment) [GNO], l'un des plus ambitieux projets actuels du monde du logiciel libre.

Le but de Bonobo est de créer un ensemble de bibliothèques pour permettre la réalisation d'applications centrées documents en utilisant une architecture de composants commune. Bonobo est largement inspiré de la technologie OLE de Microsoft (Bonobo implémente toutes les interfaces proposées par OLE). Cependant, les composants sont construits par dessus une interface CORBA [COR02].

3.5.3 Les travaux du consortium W3C

Dans le consortium W3C, le modèle objet de document (DOM) (cf.§.2.4.2) peut être considéré comme le point d'orgue de la création d'applications centrées documents.

Avec l'introduction du concept des namespaces (cf.§.2.3), les groupes de travail du W3C ont fait un important pas pour répondre aux besoins de création des documents composites. Néanmoins, Au niveau des traitements, la composition est ad hoc. C'est au développeur de l'application d'en définir la technique de traitement des différentes parties du document composite.

Afin de combler cette lacune, une activité du W3C portant sur la composition des formateurs XML est proposée dans [DFK01]. Cette proposition est très encourageante étant donné qu'elle vise à étudier les types des interactions qui peuvent s'établir entre les formateurs coopérants pour formater le document composite. Cette proposition identifie les types de données que doivent échangées les applications et leurs extensions (les plug-ins). Ainsi, des besoins en termes de formatage, de style, d'affichage et de gestion de mémoire ont été identifiés. Hélas, cette étude est suspendue actuellement.

3.5.4 Synthèse sur les documents composites

Le modèle OLE - tout comme le modèle Bonobo - tente de porter une solution au problème de traitement de documents composites en se basant sur une architecture à composant. Cependant, aucune de ces technologies n'est censée être employée comme un modèle général pour le couplage des applications traitant des documents composites. En effet, dans ces modèles, les applications conteneurs (celles qui gèrent les documents composites) ont une visibilté très restreinte des applications traitant les documents inclus. La communication entre ces applicattions se limite aux données nécessaires à l'affichage de documents inclus dans l'espace d'affichage du document composite.

3.6 Synthèse

La programmation orientée composant semble être une voie prometteuse du génie logiciel pour réduire les coûts de développement, d'évolution et de maintenance des logiciels. En effet, elle augmente les opportunités de réutilisation en définissant une nouvelle façon de structurer les applications. Celles-ci sont construites à travers un processus compositionnel.

L'état de l'art dresser dans ce chapitre est loin d'être exhaustif pour cerner tous les concepts de ce paradigme. Néanmoins, cette étude révèle que les modèles de composants « semblables »sont en émergence et aucune solution définitive ne se dégage actuellement.

L'exploitation de l'approche composant dans le domaine des documents multimédia pourrait constituer une finalité trés intéressante. Pourtant, il existe peu de travaux de recherche - si on fait abstraction des travaux sur la composition des applications centrées documents - qui invoquent cette perspective.

Au terme de cette étude, nous pouvons dégager certains besoins dont nous nous somme efforcé à prendre en considération lors de la proposition du système de couplage des formateurs :

- Comment définir le protocole de coopération entre les formateurs ?
- Comment définir les interfaces des formateurs ? Les interfaces des formateurs à coupler doivent être indépendante pour permettre de faire abstraction des environnements applicatifs,
- Comment assurer la cohérence des données échangées entre les formateurs ?

Deuxième partie

Contribution

Chapitre 4

Une architecture pour le couplage de formateurs

4.1 Introduction

Partant de l'état de l'art effectué dans la première partie, nous présentons dans ce chapitre notre contribution. Le but de ce chapitre est de donner un aperçu général de la solution afin de faciliter la lecture du reste de ce rapport.

Nous faisons dans un premier temps un point sur l'état de l'art, puis nous présentons le langage XEF et enfin nous donnons une vue générale de notre propos.

4.2 Synthèse de l'état de l'art

Au terme de l'étude théorique effectuée dans la première partie, nous voulons mettre l'accent sur quelques points qui nous paraissent essentiels et nous allons retrouver dans la spécification de notre projet :

- D'une part, nous avons identifié trois techniques permettant de faire le couplage des langages basés sur XML. Les espaces de nommage permettent de créer des documents composites. Ils fournissent un contexte aux éléments et ils permettent ainsi de leur attacher une sémantique particulière en fonction de leur provenance.
  Dans la technique de transformation, le processus de couplage consiste à convertir le document source en un autre document contenant les éléments des deux langages.
  L'approche mise en œuvre par la technique des langages de sélection est différente, elle consiste à utiliser un langage de sélection pour faire le lien entre les éléments des deux langages.
- D'autre part, l'étude du concept de la composition à travers le paradigme de la programmation par composant, nous a permis d'éclaircir plusieurs questions sur la façon de produire des applications par composition. Cette

étude nous a permis notamment de mettre l'accent sur le rôle des interfaces des composants dans cette composition.
Cependant, cette technique n'est jamais adoptée pour la composition des formateurs XML. En effet, nous avons vu qu'il existe un certain nombre de travaux pour la composition des documents électroniques, mais que le processus de conception des applications associées est souvent un processus ad hoc laissé à la charge du développeur de l'application.

4.3 Contexte de travail

Notre contexte de travail est le langage XEF issu des travaux de Frédéric Bes pendant sa thèse [BR02, BR02+].

Partant des limites des langages de présentation soulevés par l'état de l'art (cf.§.1.3.2.4), le but est d'éviter à un système de présentation d'entrer en situation d'échec. Pour ce faire :

- Le modèle de présentation doit permettre à l'auteur d'exprimer les propriétés souhaitées,
- Le formateur doit produire la solution la plus fidèle possible au document spécifié.

4.3.1 Motivations

Les travaux menés dans [BR02, BR02+] sont motivés par les besoins :

- *d'augmentation du pouvoir d'expression* : la première attente est de pouvoir donner le plus possible à l'auteur la possibilité d'exprimer ce qu'il veut obtenir. En effet, comme nous avons vu dans le premier chapitre, les langages de présentation actuels ne couvrent qu'une partie des besoins des auteurs. Le but est d'augmenter le pouvoir d'expression de ces langages en donnant à l'auteur plus de *flexibilité* et de *contrôle* dans la spécification des pésentations,
- *d'amélioration des traitements* : la deuxième attente est celle de rendre les formateurs plus contrôlables et plus ouverts, tout en respectant la spécification des langages.

4.3.2 Description du langage

Le langage XEF définit trois éléments fondamentaux qui permettent d'ajouter à un langage de présentation des fonctions de contrôle de son formatage. Ces éléments ne sont pas autonomes et dépendent les uns des autres : ce sont *les niveaux de priorité*, *les propriétés abstraites et globales* et *les techniques de repli*.

Notre expérience avec le langage XEF, nous a amené à spécifier de nouveux éléments. Ces éléments sont uilisés pour la sélection et la transformation. La grammaire du XEF est donnée dans l'annexe 2.

4.3.2.1 Les niveaux de priorité

L'élément *priorités* donne à des éléments ou des relations différents niveaux d'importance. Par exemple il permet d'indiquer au formateur, quand il est en cas d'échec, l'ordre dans lequel effectuer les traitements alternatifs.

XEF permet de déclarer un ou plusieurs ordres de priorités sur un même ensemble d'éléments permettant ainsi d'attacher chaque ordre à une opération particulière (par exemple, un ordre pour le temporel et un autre ordre pour le spatial).

En guise d'exemple de priorités, nous donnons dans la figure FIG.4.1 un élément *priorités* qui regroupe des priorités affectées aux différentes parties du document présenté dans la figure FIG.5.1 (cf.§.5.2.1). La priorité (p4) est définie à partir des deux priorités (p1) et (p3). La priorité (p1) précise les préférences sur les résumés de scènes de la présentation (les résumés du film et des interviews de l'acteur et du réalisateur). Elle est définie à partir d'une autre priorité qui établit un ordre croissant entre les éléments *ResumeActeur1* et *ResumeRealisateur1*. La priorité (p3) donne un ordre sur les éléments *VideoActeur1* et *VideoRealisateur1*.

Comme le montre cet exemple, les priorités peuvent être imbriquées. De plus, l'exemple montre que les priorités peuvent être définies aussi bien sur des relations (cf.lignes 2-9) que sur des éléments de base (les média).

```
1. <priorités>
2.  <ordre id= "p4">
3.   <ordre id= "p1">
4.    <element ref= "RésuméFilm1">
5.    <ordre id= "p2">
6.     <element ref= "RésuméActeur1">
7.     <element ref= "RésuméRéalisateur1">
8.    < /ordre>
9.   < /ordre>
10.  <ordre id= "p3">
11.   <element ref= "VideoActeur1">
12.   <element ref= "VideoRéalisateur1">
13.  < /ordre>
14. < /ordre>
15. < /priorités>
```

Fig. 4.1 – Exemple de priorités XEF

4.3.2.2 Propriétés abstraites et globales

Elles permettent de spécifier des contraintes partielles ou globales sur le document à formater. Ces contraintes permettent de prendre une décision par rapport à la sémantique du document au travers d'un ou plusieurs critères dont la valeur doit être optimisée. Dans l'exemple de la figure FIG.5.1, l'auteur peut vouloir exprimer qu'il ne veut pas plus d'un interview pour chaque film, que la durée de la présentation ne dépasse pas 12 mn et que les zones spatiales soient équilibrées.

4.3.2.3 Techniques de repli

D'une manière générale, les techniques de repli permettent de proposer des pistes aux formateurs quand ils sont en situation d'échec. Dans le langage XEF, les replis sont des spécifications de formatage qui portent sur une relation et ses éléments ou directement sur un élément d'une relation. Ces spécifications peuvent être classées en deux types : *les alternatives* et *les contrôles*.

4.3.2.3.1 les alternatives

La majorité des langages de présentation proposent des éléments pour exprimer les alternatives. C'est notamment le cas de XSL-FO, SMIL et Madeus (cf.§.1.3.2.4). Un point commun entre ces langages est que le choix d'une alternative est basé sur la valeur d'un attribut. Les alternatives telles que spécifiées dans le langage XEF permettent de spécifier un choix non pas par rapport à la valeur d'un attribut mais suite à l'échec du formatage. Elles imposent une redéfinition complète des nœuds à remplacer.

Dans l'exemple de la figure FIG.4.2, une alternative *alt(rel_seq)* est placée sur l'élément *par* qui se compose d'une vidéo et d'un texte. En cas d'échec du formatage, cet élément sera remplacé par une séquence *rel_seq* (cf.ligne7-10). Ce dernier sera formaté et inclus dans le document uniquement si le formatage de l'élément *rel_par* devait échouer.

4.3.2.3.2 Les contrôles

Alors qu'une alternative impose une redéfinition complète des nœuds à remplacer, un contrôle modifie seulement la spécification de formatage de la relation ou de l'élément sur laquelle elle s'applique. XEF définit plusieurs types de contrôles :

- Le contrôle *préférentiel* est le contrôle par défaut qui force les éléments à prendre la valeur donnée par l'auteur,
- Le contrôle *flexible* spécifie que le formatage de la relation pourra rendre flexible les éléments qui le permettent,
- Le contrôle *suppression* permet la suppression des éléments,
- Les contrôles *réduction* et *extension* permettent la réduction ou l'extension des éléments dans les limites des domaines de définition,

```
1. ...
2. <par id="rel_par" alt(rel_seq)>
3.  <video id="BandeAnnonce" src= "film.mpeg" region="video" />
4.  <texte id="text" src= "discription.rtf" region="description" />
5. < /par>
6. ...
7. <seq id="rel_seq" type=="alt" >
8.  <video id="BandeAnnonce" src= "film.mpeg" region="video" />
9.  <texte id="text" src= "discription.txt" region="video" />
10. < /seq>
11. ...
```

FIG. 4.2 – Exemple d'une alternative XEF

– Enfin, le contrôle *fixe* permet de geler les valeurs des éléments sur lesquels il est placé.

Ces contrôles sont appliqués sur les éléments et les relations suivant l'ordre donné par les priorités.

Dans l'exemple présenté dans la figure FIG.4.3, deux contrôles sont placés sur la séquence *Premier_Film* : *réduction (p1)* et *suppression (p2)*. Le premier contrôle spécifie que tous les éléments de la relation *Premier_Film* vont, en respectant l'ordre donné par la priorité (p1), être réduits si nécessaire. Puis, si cela ne suffit pas, ils vont, en respectant l'ordre donné par la priorité (p3), être remplacés par leur alternative respective. Enfin, si le formateur est toujours en cas d'échec, le contrôle *suppression (p2)* permet la suppression des éléments en respectant la priorité (p2).

```
1. <seq id="Premier_Film" repli="réduire (p1), alt(p3), suppression (p2)"
   >
2. ...
3. < /seq>
```

FIG. 4.3 – Exemple de contrôle XEF

4.3.2.4 Les nouveaux éléments

Durant ce stage, nous avons constaté le besoin d'un mécanisme permettant de spécifier les mêmes contrôles sur plusieurs éléments du document source. En effet, il est fréquent que le document source contienne des éléments ou des relations qui exprime les mêmes scénarios sur des données différentes. Dans le document *ProgrammeCinémas* de la figure FIG.5.1, quelque soit le film, le scénario

```
1. <foreach select="//body/par/seq/par">
2.  <variable name="parnb" select="position(.)"/>
3.  <groupe id="par1">
4.   <parTemporal>
5.    <attribute name= "id">VideoDescription<value-of select="parnb"/>
6.    < /attribute>
7.    <repli>seqTp¡value-of select="parnb"/ >< /repli>
8.    ....
9.   < /parTemporal>
10.  <seqTemporal alternative="oui">
11.   ...
12.  < /seqTemporal>
13.  <groupe id="par1">
14. < /foreach>
```

FIG. 4.4 – Exemple d'un contrôle XEF *foreach*

de la présentation est le même (résumé du film, résumé de l'interview de l'acteur et enfin résumé de l'interview du réalisateur). Nous avons donc proposé des éléments permettants à l'auteur de d'exprimer deux types de fonctionnalités :

- *La sélection* : ces éléments sont spécifiés pour répondre au besoin d'avoir un mécanisme facilitant l'opération de désignation. Ainsi, nous avons définis l'élément *ref* et l'attribut *select* pour la sélection des nœuds simple et l'élément *groupe* pour la désignation de plusieurs éléments,
- *les traitements repetitifs* :l'élément *foreach* permet d'appliquer les mêmes contrôles à l'ensemble d'éléments spécifiés par une expression de chemin. Dans l'exemple de la figure FIG.4.4, un contrôle *alternative* est spécifié pour chaque élément *par* ayant le chemin *//body/par/seq/par* dans le document source.

4.3.3 Synthèse sur le langage XEF

De part son niveau d'abstraction intéressant et sa capacité d'expressivité, le langage XEF paraît répondre à certaines limitations des langages de présentation relevées dans le premier chapitre. Ce niveau d'expression est obtenu d'une part par la définition des priorités sur les nœuds du document et d'autre part par l'ajout, aux relations classiques de présentation, de critères globaux d'optimisation et de replis. Ces éléments sont indépendants de n'importe quel langage de formatage existant. Ils peuvent être intégrés dans ces langages dans l'optique d'un meilleur contrôle de formatage.

4.4 Proposition

Disposant du langage XEF qui offre des éléments de contrôle indépendants des langages de formatage, nous souhaitons avoir un système de génération des présentations multimédia intégrant aussi bien le langage XEF que son traitement dans les formateurs existants tout en gardant le plus d'indépendance possible entre le formateur XEF et le formateur existant. L'intérêt d'une telle démarche est triple :

- Premièrement, on réduit le travail de l'auteur. Pour ce faire, il nous semble intéressant de permettre à l'auteur de spécifier les éléments de contrôle du langage XEF dans un document séparé du document source à contrôler. En effet, cette approche permet à l'auteur non seulement de contrôler le formatage de nouveaux documents mais de pouvoir contrôler le formatage des documents existants sans qu'il soit obligé de rééditer ces documents,
- Deuxièmement, on tire profit des formateurs existants. En effet, l'évolution de la technologie de traitement des documents électroniques est marquée par l'apparition de formateurs performants. Notre ambition n'est pas de remplacer ces formateurs mais de proposer des solutions pour palier leurs défauts en intégrant le formateur XEF.
- Troisièmement, on réduit le travail du développeur de l'application[1]. Celui-ci n'a plus à se plonger dans le code des deux formateurs pour faire le couplage, mais seulement à spécifier les points d'interconnexion des formateurs.

4.4.1 Processus de formatage en utilisant XEF

Le choix que nous avons adopté nous a conduit à définir une architecture générique pour le processus de génération d'un document contrôlé à travers le langage XEF. La figure FIG.4.5 illustre l'architecture de notre système de formatage. Les différentes parties de cette architecture sont :

- *Le document source* qui contient la description spatio-temporelle de la présentation. Aucune hypothèse n'est faite sur le langage de présentation utilisé,
- *Le document de contrôle* qui contient les éléments de contrôles XEF et spécifie les contrôles à appliquer sur le document source. Nous exposons la technique du couplage des deux documents au chapitre 5,
- *Le formateur* qui permet, à partir du document source et du document de contrôle, de générer les structures nécessaires pour être directement exécutables. L'architecture du formateur occupe une place particulière qui est au cœur de l'architecture générale de notre système. Afin de tirer profit des formateurs existants et dans le but de permettre une meilleure réutilisation de ces formateurs, nous proposons de construire ce formateur à partir du formateur du langage source et du formateur du langage XEF. La construction du formateur à partir de formateurs existants nécessite une étape d'assemblage. Le couplage du formateur XEF avec le formateur

[1]Nous entendons par le développeur de l'application, la personne chargée de faire le couplage du formateur XEF avec le formateur à contrôler.

du langage source se traduit par l'établissement d'une jonction entre ces deux formateurs. Cette jonction s'établit au niveau d'un ensemble d'interfaces de connexion. Pour rendre l'architecture du système indépendante des formateurs utilisés, nous devons spécifier des interfaces génériques. Nous décrivons en détails la spécification du système de composition au chapitre 6,

– *Le module d'exécution* qui se charge de l'interprétation des informations résultantes du formatage. Il s'agit du module d'exécution du langage source une fois formaté.

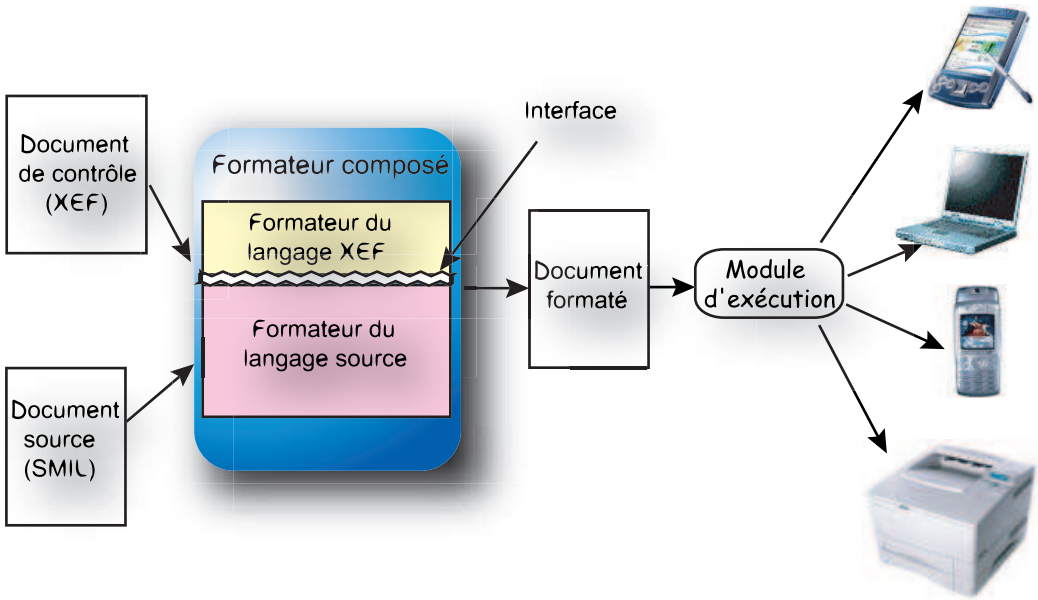

FIG. 4.5 – Architecture générale du système de contrôle de formatage

Chapitre 5

Couplage du langage XEF avec des langages de formatage existants

5.1 Introduction

Dans ce chapitre, nous présentons notre proposition pour le couplage au niveau des langages. L'état de l'art dressé dans le chapitre 2, nous a permis de distinguer plusieurs techniques pour coupler deux langages de balisage. Les solutions proposées s'en inspirent et tentent d'en combler les lacunes.

Nous proposons quatre techniques pour coupler le langage XEF avec un langage de formatage existant. Pour chacune de ces solutions nous donnons le principe, son application sur un exemple et enfin nous discutons les apports et les limites par rapport aux besoins du couplage du langage XEF avec d'autres langages.

5.2 Cas d'étude

5.2.1 Description

Tout au long de ce chapitre, nous illustrons nos propositions à travers le document présenté dans la figure FIG.5.1. Ce document que nous appelons *ProgrammeCinemas*, permet la présentation des programmes des cinémas d'une ville.

---

1. <smil>
2. <head>
3. <layout>
4. <root-layout height="442" width="770"/>

```
5.    <region id="cinema" top="15" left="200" height="45" width="370" />
6.    <region id="title" top="392" left="200" height="35" width="370" />
7.    <region id="video" top="80" left="20" height="292" width="350" />
8.    <region id="description" top="80" left="390" height="292" width-
    ="385" />
9.    < /layout>
10.  < /head>
11.  <body>
12.   <seq>
13.    <par id="ParCinéma1">
14.     <text id="NomCinéma" region="cinema" src="Ville1.html" />
15.     <seq id="SeqCinéma1">
16.      <par id="RésuméFilm1">
17.       <text id="TitreFilm" region="title" src="Film1.html" />
18.       <par id="ParFilm">
19.        <video id="BandeAnnonce" region="video" src="Film1.mpeg" />
20.        <text id="FilmDescr" region="description" src="Discription.html" />
21.       < /par>
22.       <par id="RésuméActeur1">
23.        <video id ="VideoActeur1" region="video" src="Dicaprio1.mov" />
24.        <text id="ActeurDescr1" region="description" src="Dicaprio1.html" />
25.       < /par>
26.       <par id="RésuméRéalisateur1">
27.        <video id="VideoRéalisateur1" region="video" src="Cam1.mov" />
28.        <text id="RéalisateurDescr1" region="description" src="Cam1.html" />
29.       < /par>
30.      < /par>
31.      <par id="ParFilm2">...< /par>
32.     < /seq>
33.    < /par>
34.    <par id="ParCinéma2">...</par>
35.   ...
36.   < /seq>
37.  < /body>
38. < /smil>
```

FIG. 5.1 – Exemple de document *ProgrammeCinémas*

Le document est décrit dans le langage SMIL 2.0. L'élément *layout* défini dans l'en-tête *head*, spécifie le positionnement spatial des éléments utilisés dans la partie *body*. La fenêtre mère dont les attributs de présentation sont définis sur l'élément englobant *root-layout* peut être subdivisée en autant de régions de présentation que voulu. Dans notre cas, nous avons défini quatre régions (cf.FIG.1.4).

L'élément *body* contient le scénario spatio-temporel que l'on veut construire. Le scénario temporel (cf.FIG.5.2) consiste à jouer deux éléments en parallèle : le nom du cinéma et une séquence qui décrit le programme de ce cinéma. Celui-ci correspond à une séquence de résumés des films présentés dans ce cinéma. Chaque film est représenté par deux éléments joués en même temps : le titre et un résumé. Le résumé d'un film est une scène composée de trois sous-scènes. La première consiste à présenter une bande annonce du film en même temps qu'une description textuelle (nom du réalisateur, noms des acteurs, des producteurs,...). La deuxième sous-scène présente une interview avec l'acteur principal du film et une description textuelle de son rôle dans le film. Enfin, la dernière sous-scène consiste à présenter en même temps une interview avec le réalisateur et un texte descriptif.

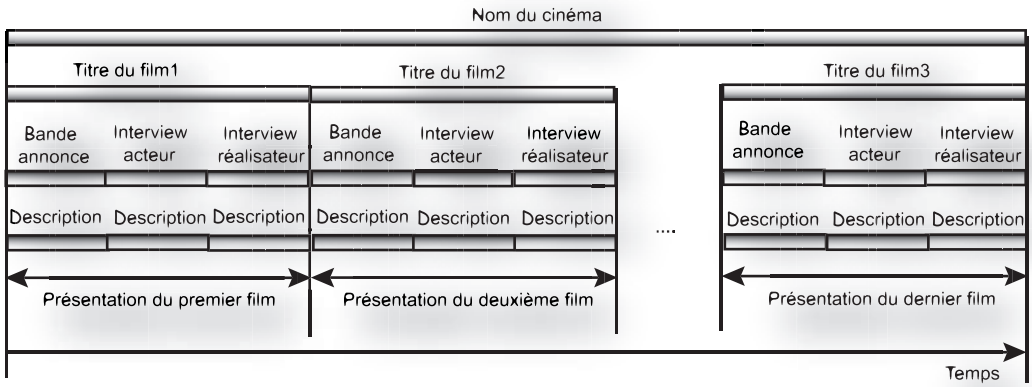

FIG. 5.2 – Scénario temporel du document *ProgrammeCinémas* pour un grand écran

5.2.2 Besoins de présentation

5.2.2.1 Principe

En utilisant les éléments de contrôle offerts par le langage XEF, nous cherchons à ajouter une sémantique de présentation plus précise sur les opérateurs élémentaires *par* et *seq*. Le but est d'orienter le programme de formatage durant le calcul des valeurs des propriétés de présentation pour éviter les cas d'echec. En

effet, les éléments temporaux SMIL ont une sémantique très précise et les descriptions spatiales sont très rigides. En revanche, il ne founit aucun mécanisme pour contrôler plus finement le formatage d'où les cas d'échec. Par exemple :

- Le langage SMIL ne fait pas abstraction de terminaux de sortie. C'est à l'auteur de spécifier la présentation appropriées à chaque type de terminaux,
- SMIL ne permet pas une spécification fine des propriétés. Par exemple, il ne donne pas le moyen pour équilibrer les régions, l'auteur doit préciser finement les propriétés des régions,
- SMIL ne permet non plus la spécification des propriétés globales qui vont permettre d'adapter le formatage du document au temps dont dispose le lecteur.

A partir de ce document SMIL nous voulons avoir plus de flexibilité dans la spécification du sénario spatio-temporel. Pour le faire, nous rajoutons :

- du contrôle sur les zones spatiales en proposant des alternatives de placement,
- du contrôle sur l'organisation temporelle en définissant des priorités et des règles de repli sur les éléments temporels et sur les média : les bandes annonces, les interviews,...

5.2.2.2 Exemple de besoins

Plusieurs présentations peuvent être définies pour le document *ProgrammeCinémas*. Suivant la taille d'écran, par exemple, on peut imaginer deux présentations différentes :

- Une présentation destinée aux grands écrans : dans ce cas, le scénario spatio-temporel est celui décrit ci-dessus (cf.FIG.5.1),
- Une présentation destinée aux média de type PDA : compte tenu de la taille d'écran relativement petite dans ce type de média, l'adoption de l'emplacement spatial de la présentation destinée aux grands écrans, rend le document inexploitable, il faut utiliser les ascenseurs pour naviguer dans la présentation (cf.FIG.5.3). D'un point de vue ergonomique l'utilisation des ascenseurs n'est pas préférable. Une autre présentation spatiale est donc nécessaire.

Nous proposons par exemple, de garder l'emplacement du titre du film et du nom du cinéma tels qu'ils sont spécifiés dans la première présentation et de présenter la vidéo et le texte descriptif de chaque résumé (le film, l'interview de l'acteur et l'interview du réalisateur) en séquence et dans deux régions spatiales superposées et centrées. Nous souhaitons aussi, que les régions spatiales des quatre éléments : nom du cinéma, titre du film, la vidéo et le texte descriptif, soient équilibrées verticalement par rapport à la fenêtre mère. En plus, nous aimerions que les zones de présentation de la vidéo et du texte descriptif soient équilibrées horizontalement par rapport à la fenêtre mère. Hélas, SMIL ne sait gérer que des placements spatiaux absolus.

FIG. 5.3 – Organisation spatiale du document *ProgrammeCinémas* sur un PDA avant le contrôle

5.2.2.3 Spécification des besoins avec le langage XEF

Pour combler les lacunes constatées, nous rajoutons, en utilisant le langage XEF, des relations d'alignement des régions. Même chose pour le scénario temporel, au lieu de devoir gérer cet aspect à la main, l'ajout d'un attribut de contrôle précise que la séquence est équilibrée laisse ce travail au formateur.

Nous définissons des degrés de priorités sur les média et sur les éléments temporels : le résumé du film est plus important que l'interview de l'acteur et du réalisateur (ces priorités correspondent à la spécification donnée dans la figure FIG.4.1). Nous spécifions également des règles de repli pour le formatage : si le scénario est trop long, d'abord réduire le résumé d'interview du réalisateur, ensuite celui de l'acteur et enfin le résumé du film. Si cela ne suffit pas, supprimer progressivement les résumés des interviews du réalisateur et de l'acteur. La spécification XEF permettant de décrire ces contrôles est donnée dans la figure FIG.5.4 (les priorités $p1$ et $p2$ sont celles données dans la figure FIG.4.2).

<par id=“ParCinéma1” repli=”réduire(p1),suppression(p2)>

FIG. 5.4 – Contrôle XEF du document *ProgrammeCinémas*

La figure FIG.5.5 présente un exemple d'une organisation spatiale qui respecte les contraintes ci-dessus. Comme les régions vidéo et description se chevauchent, le sénario temporel doit être modifié pour permettre la présentation de la vidéo et du texte descriptif. Nous proposons de jouer le texte descriptif et la vidéo de chaque résumé (résumé du film, résumé de l'interview de l'acteur et résumé de l'interview du réalisateur) en séquence et le tout est joué en même temps que le titre du film. Toutes les sous-scènes représentant les films d'un cinéma sont jouées en parallèle avec le nom du cinéma. Ce scénario est illustré dans la figure FIG.5.6. La configuration de l'écran sur la figure FIG.5.5.A cor-

respond à l'instant t_0 de la présentation. Les éléments nom du cinéma, titre du film et la vidéo de la bande annonce sont présents. La configuration de l'écran sur la figure FIG.5.5.B présentant l'instant t_1 de la présentation, montre que les éléments nom du cinéma et titre du film sont encore présents alors que la vidéo est remplacée par le texte descriptif.

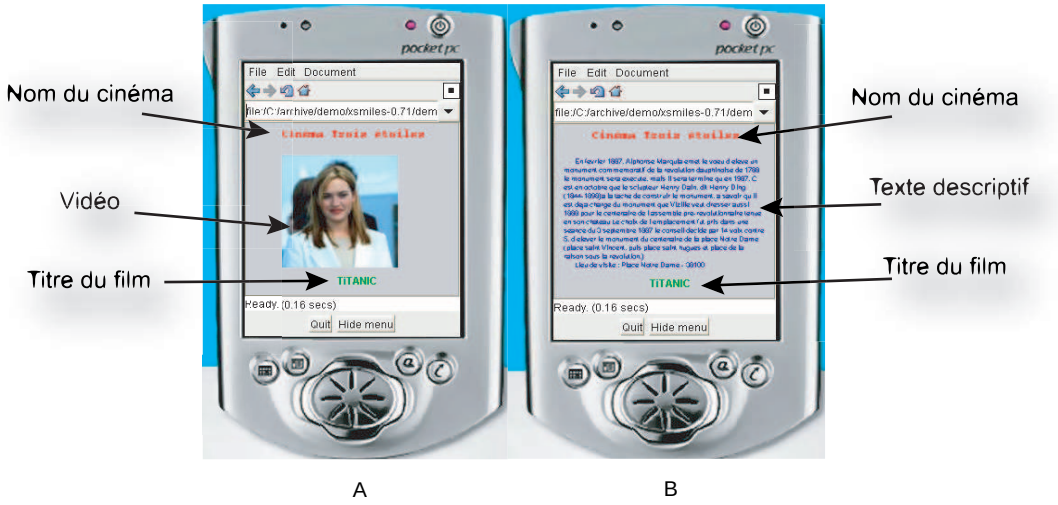

FIG. 5.5 – Organisation spatiale du document *ProgrammeCinémas* sur un PDA aprés le contrôle

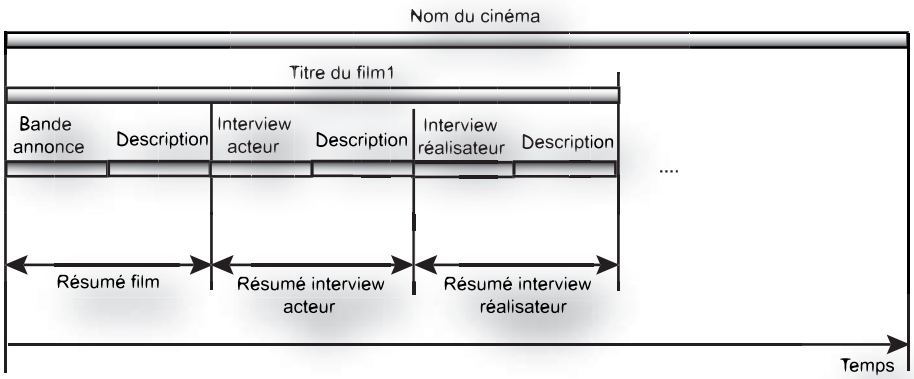

FIG. 5.6 – Exemple d'un scénario temporel du document *ProgrammeCinémas* pour un PDA

Avant de présenter quelques propositions pour le couplage du langage XEF avec d'autres langages de présentation, nous précisons dans le paragraphe suivant les besoins auxquels les techniques proposées doivent répondre.

5.3 Identification de besoins

La technique de couplage du langage XEF avec d'autres langages de présentation doit répondre à certains besoins, notamment ceux qui correspondent à la *modification* du document source. Nous entendons par modification tout traitement permettant :

- *L'insertion* : elle consiste à ajouter des nœuds (éléments, attributs ou texte). Dans l'exemple de la figure FIG.4.2, l'élément *rel_seq* est de type alternative, le formatage de cet élément consiste à ajouter à l'arbre source un élément de type *seq*,
- *La modification* : elle porte sur les valeurs des attributs ou des nœuds textuels. Par exemple, l'équilibrage d'une séquence d'éléments consiste à recalculer les valeurs des propriétés de ces éléments,
- *La suppression* : la sémantique de l'élément *suppression* consiste systématiquement à supprimer les noeuds sur lesquels il est placé.

5.4 Couplage par décoration

5.4.1 Présentation

La première technique intéressante pour coupler le langage XEF avec d'autres langages de formatage est inspirée de la technique des feuilles de style dont l'objectif est de séparer le contenu des documents de leur présentation.

Une feuille de style est composée d'un ensemble de *règles de style*. Chaque règle est composée d'un *sélecteur* et d'un *style* à appliquer aux éléments identifiés par ce sélecteur. Comme nous le mentionnions au chapitre 2, un certain nombre de langages de sélection ont été proposés pour mettre en œuvre un tel mécanisme.

Nous présentons dans ce paragraphe comment on peut exploiter la technique des langages de sélection pour le couplage des langages. Le but est d'utiliser un langage de sélection pour identifier les éléments du document source sur lesquel on veut rajouter des contrôles de formatage XEF.

Dans cette technique, les éléments de contrôle de formatage sont spécifiés dans un document indépendant du document source (par la suite nous désignons ce document sous le nom : *document de contrôle*).

Un document de contrôle contient un ensemble de *règles de contrôle*. Une règle de contrôle est une instruction indiquant au formateur le contrôle à effectuer sur des éléments particuliers du document source. Chaque règle de contrôle est constituée de deux parties, un *sélecteur* et un *contrôle* à appliquer aux éléments identifiés par ce sélecteur.

Le sélecteur : nous utilisons un langage de sélection comme XPath ou le

langage de sélection de CSS (cf.§.2.5) pour identifier les nœuds (les éléments, les attributs et le texte) à contrôler dans le document source.

Le contrôle : les traitements de contrôle sont spécifiés en utilisant les éléments du langage XEF.

Avec cette technique de couplage, le processus de formatage se déroule de la façon suivante : pour chacun des éléments du document source, le formateur cherche, dans le document de contrôle, s'il y a des règles de contrôles associées. Si c'est le cas, la partie contrôle de chaque règle de contrôle est utilisée pour décorer le nœud de cet élément. La figure FIG.5.7 illustre le processus de génération d'une présentation à partir d'un document source SMIL et d'un document de contrôle XEF. Au moment de l'analyse lexicale (parsing), le processeur SMIL cherche pour chaque élément SMIL, des règles de contrôle dans le document XEF et décore les nœuds correspondant dans l'arbre DOM en utilisant la partie contrôle de chaque règle.

Pour attacher le document de contrôle au document source, nous proposons deux solutions :

- Le lien se fait dans le document source en utilisant par exemple, l'instruction de traitement proposée par le consortium W3C dans la recommandation [Cla99] :

  $<?XML\text{-}stylesheet\ href = "\text{URL\_ControlSheet}"\ type = "\text{MIME\_Type}" >$

- Le lien entre le document cible et le document de contrôle peut être aussi spécifié dans un document tierce.

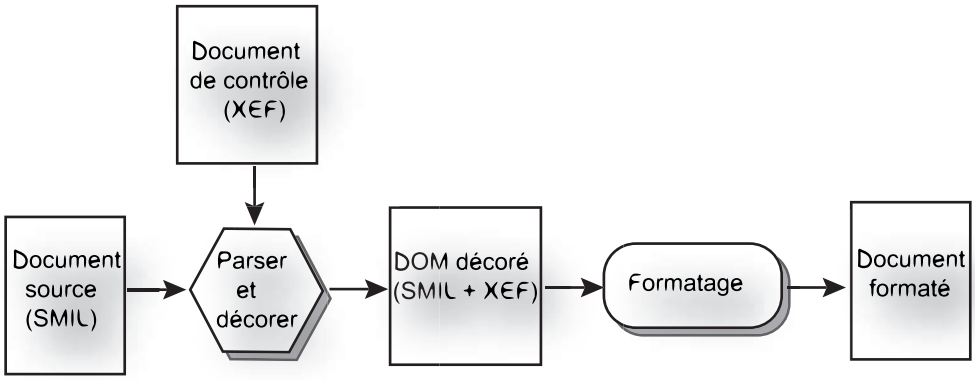

FIG. 5.7 – Technique de couplage des langages par décoration

5.4.2 Exemple

Nous présentons dans la figure FIG.5.8, une règle de contrôle permettant d'illustrer le principe des sélecteurs/contrôles pour faire le couplage du langage XEF avec les éléments du document SMIL présenté dans la figure FIG.5.1. Le

sélecteur */SMIL/body/seq/par/seq* permet d'identifier tous les éléments *seq* fils d'un élément *par*, lui même fils d'un élément *seq* et de leur appliquer deux contrôles : *réduction* et *suppression*.

1. /SMIL/body/seq/par/seq : {reply="réduire(p1), suppression(p2)"}
2. /SMIL/body/seq : {équilibrage="oui"}

FIG. 5.8 – Exemple de couplage en utlisant les langages de sélection

5.4.3 Discussion

Une limite majeure de cette technique est que les langages de sélection ne permettent que la décoration (à la CSS) de l'arbre du document source, ils ne permettent pas d'ajouter des éléments à l'arbre alors que concrètement, l'ajout d'un contrôle consiste à insérer dans l'arbre du document des nœuds autour d'autres nœuds ou à ajouter des attributs aux éléments du document.

Dans l'exemple *ProgrammeCinémas*, nous voulons que la présentation de la vidéo et du texte descriptif associé (pour chaque résumé : le film, les interviews de l'acteur et du réalisateur) soit en séquence sur un PDA. Concrètement, ceci revient à rajouter à l'arbre du document source une séquence de type *alternative* pour chaque élément *par* définissant un résumé. En revanche, les langages de sélection ne permettent pas à eux seuls d'effectuer ces modifications sur l'arbre source.

Un certain degré de souplesse a été offert par quelques langages de sélection comme le langage de sélection de CSS. Ce dernier permet à l'auteur de spécifier le style et l'emplacement d'un contenu généré au moyen des pseudo éléments : *before* et *after*. Comme leurs noms l'indiquent, ceux-ci précisent l'emplacement du contenu avant ou après celui d'un élément de l'arbre du document. La propriété *content*, utilisée en conjonction avec eux, spécifie la nature de ce qui est inséré. Dans l'exemple ci-dessous (cf.FIG.5.9), il est spécifié pour les éléments *text*, qu'avant le contenu de l'élément, le texte statique *Prix :* soit affiché et après le contenu de l'élément, le texte *EUR*.

1. text : before {content : "Prix : "}
2. text : after {content : "EUR "}

FIG. 5.9 – Exemple d'éléments CSS

Cependant cette souplesse reste insuffisante. En effet, ces éléments ne permettent que l'ajout du texte (que se soit statique ou dynamique) alors que l'ajout d'un contrôle XEF entraîne des modifications sur des nœuds divers (texte, attribut et élément).

La solution serait d'augmenter l'expressivité du langage de sélection pour permettre l'insertion des nœuds de types divers (éléments, attributs, texte) dans

l'arbre du document source mais à priori, les techniques de transformation sont mieux adaptées. En effet, le but premier de ces techniques est de permettre la modification (l'ajout, la suppression et la modification des nœuds) de l'arbre source.

5.5 Couplage par transformation

5.5.1 Rappel sur les langages de transformation

Comme nous le mentionnons au chapitre 2, plusieurs langages de transformation ont été proposés. Ils peuvent être classés en trois catégories [Vil02] :

- Les langages de programmation classique de type : Java, Perl et C. ils sont expressifs mais ils sont difficiles à utiliser ;
- Les langages intégrant des fonctions de transformation de type : XML Script [XMLS]. Bien que ces langages facilitent la transformation, se sont des langages impératifs et par conséquent leur utilisation reste relativement complexe ;
- Enfin, la troisième catégorie à laquelle nous nous intéressons puisqu'elle regroupe les langages dédiés exclusivement à la transformation des documents de balisage (SGML ou XML) comme XSLT.

Un point partagé entre les langages dédiés à la transformation est qu'ils génèrent tous un nouveau document après la transformation. Selon le langage dans lequel est spécifié ce document, nous distinguons deux cas :

- Le document cible est spécifié dans un seul langage, qui éventuellemenet peut être différent du langage source,
- Le document cible est un document composite contenant des éléments spécifiés dans plusieurs vocabulaires XML.

5.5.2 Couplage par transformation vers un langage cible

5.5.2.1 Présentation

La deuxième technique que nous proposons consiste à transformer le document source en un document cible spécifié dans un langage de balisage existant, bien maîtrisé et pour lequel des logiciels de formatage sont disponibles.

D'un point de vue langagier, le document source est converti en un document cible spécifié dans un langage de balisage cible. Ce dernier peut être différent du langage dans lequel est spécifié le document source mais il doit être au moins autant expressif que le langage source et supportant les éléments de contrôle du langage XEF. D'un point de vue syntaxique, cette transformation, permet de réorganiser les éléments du document source en les filtrant, en les déplaçant, en les renommant, ou en générant de nouveaux éléments pour les exprimer dans le langage cible. Ensuite, le document cible est formaté en utilisant un formateur du langage cible.

Dans cette technique, le scénario spatio-temporel est spécifié dans le document source alors que les éléments de contrôle de formatage sont spécifiés dans un document indépendant, par contre, il est tout à fait possible de les incor-

porer dans la feuille de transformation. On peut en effet, indépendamment du langage de contrôle, transformer tout document SMIL par exemple, en Madeus et la création d'une nouvelle feuille de transformation pour un document donné reviendrait à ajouter les éléments de contrôle avec des règles plus précises.

Nous avons présenté dans la figure FIG.5.10, le processus de production d'une présentation contrôlée en utilisant la technique de transformation pour coupler les éléments de contrôle XEF à un document source. Le document source décrit dans le langage SMIL est transformé, en utilisant une feuille de transformation XSLT et le document XEF, en un document cible spécifié dans le langage Madeus+. Le document Madeus+ ainsi créé est formaté en utilisant un formateur Madeus+. Dans cet exemple, nous avons fait l'hypothèse que le langage Madeus+ supporte les éléments de contrôle XEF (faut-il rappeler qu'il n'existe pas un langage de formatage qui supporte les éléments XEF).

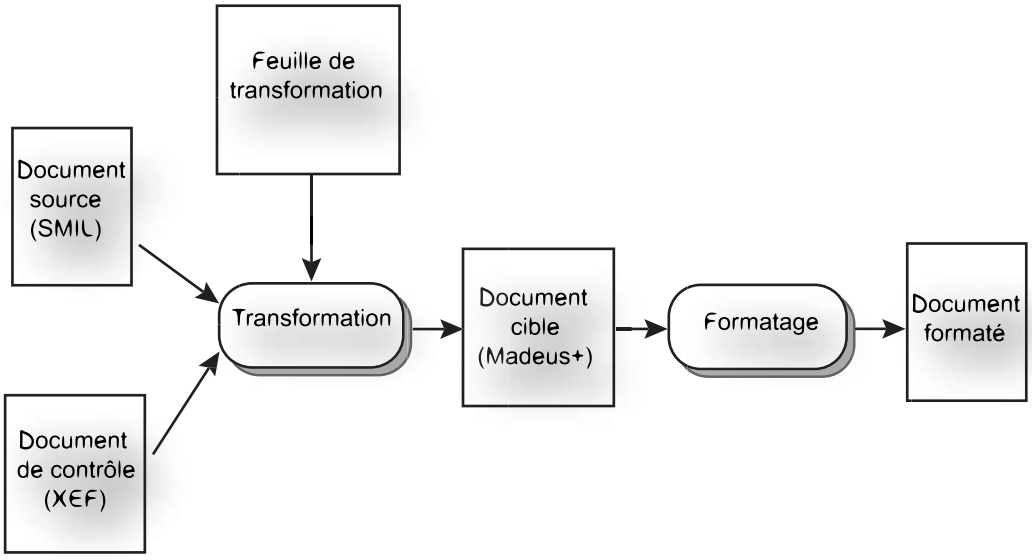

FIG. 5.10 – Couplage par transformation vers un langage cible

5.5.2.2 Exemple

Supposant qu'il existe un langage de balisage cible Madeus+ qui supporte les éléments de contrôle du langage XEF, ainsi que les éléments du langage SMIL. Nous donnons dans la figure FIG.5.11, un extrait d'une feuille de transformation permettant de convertir le document SMIL donné dans la figure FIG.5.1 en un document cible adapté à la présentation sur un écran d'un PDA, en respectant les préférences que nous avons annoncées précédemment. Le document cible est spécifié dans le langage Madeus+. Dans cet exemple, les éléments de contrôle du langage XEF sont incorporés dans la feuille de transformation.

Au début du processus de transformation, le processeur de transformation

recherche une règle (un élément de type *template*) s'appliquant à la racine du document source (dans l'exemple la racine est l'élément SMIL). Pour chaque règle, le processeur teste si le motif (un motif caractérisant un ensemble de nœuds), qui est représenté par la valeur de l'attribut *match*, correspond à l'élément SMIL. La règle trouvée est exécutée avec comme nœud source courant l'élément SMIL (lignes 2-11 de la feuille de transformation). Cette règle est composée d'une suite d'instructions qui sont exécutées séquentiellement. Chaque instruction a une action particulière. Dans l'exemple, les éléments *smil* et *head* sont copiés dans le document cible. Le contenu de l'en-tête est généré grâce à l'instruction *apply-templates* (ligne 6). Alors que le contenu de l'instruction *apply-templates* de la ligne 8 génère le contenu du body du document cible.

L'exécution de l'instruction apply-templates de la ligne 6 consiste dans un premier temps à sélectionner un ensemble de nœuds dans le document source en utilisant l'expression XPath (head/layout). Ensuite pour chacun de ces nœuds, les règles correspondant sont recherchées dans la feuille de transformation. Dans l'exemple, cette instruction sélectionne l'élément unique *layout* à partir du nœud source courant *smil*, c'est-à-dire l'élément des lignes 3-9 du document source. La règle de la feuille de transformation à appliquer est celle des lignes 15-34.

L'exécution de cette règle commence par la création de l'élément *root-layout* dans l'arbre cible. Ensuite, l'élément *seq-spatial* est ajouté à l'arbre cible. L'élément *équilibrage* est créé dans l'arbre cible comme un fils de l'élément *seq_spatial* (ligne 17-33) avec comme valeur *equilibre_V_father* qui permet d'équilibrer verticalement les quatre placements spatiaux.

L'instruction des lignes 20-27 insère dans le document cible un autre élément *seq_spatial* qui permet, entre autres, d'ajouter une relation de *repli*. Cette relation propose des pistes au formateur quand il est en situation d'échec. Elle comprend des spécifications de formatage qui portent sur l'emplacement spatial des éléments (ligne 23) aussi bien que sur le scénario temporel (ligne 24).

De la même façon, l'exécution de l'instruction de la ligne 8 commence par la sélection dans le document source des nœuds qui correspondent à l'expression XPath (body). Ensuite, la règle correspondant dans la feuille de transformation est exécutée (à partir de la ligne 35).

5.5.2.3 Discussion

L'avantage de cette technique est qu'elle permet de générer un document cible spécifié dans un langage de balisage existant. Il est ainsi possible d'utiliser un formateur du langage cible pour formater le document. Ceci permet de d'éviter le problème de couplage des formateurs.

Cependant, dans l'état actuel des langages de balisage, il n'existe aucun langage qui supporte les éléments de contrôle du langage XEF. De ce fait, pour mettre en œuvre cette technique, il faut passer par un processus d'*extension des langages*. Pour ce faire, nous proposons d'étendre le langage XEF pour supporter le langage source. Dans ce cas, il faut, pour chaque langage source, étendre le langage XEF pour qu'il supporte les éléments de ce langage. Ainsi, pour contrôler le formatage des documents sources SMIL, il faut étendre le

```
1. ....
2. <xsl :template match="smil">
3.  <xsl :message>Debut de la transformation</xsl :message>
4.  <smil>
5.   <head>
6.    <xsl :apply-templates select="head/layout"/>
7.   < /head>
8.   <xsl :apply-templates select="body"/>
9.  < /smil>
10.  <xsl :message>Fin de la transformation</xsl :message>
11. < /xsl :template>
12.  <xsl :template match="root-layout">
13.   <xsl :copy><xsl :copy-of select="@*"/></xsl :copy>
14.  < /xsl :template>
15.  <xsl :template match="layout">
16.   <xsl :apply-templates select="root-layout"/>
17.   <xef :seq_spatial id="seqSp0">
18.    <équilibrage>equilibrate_V_father</équilibrage>
19.    <xsl :copy-of select="region[@id='town'] | region[@id='title']"/>
20.     <seq_spatial id="seqSp1">
21.      <équilibrage>equilibrate_H_father</équilibrage>
22.      <repli>
23.       <relation>seqSp2</relation>
24.       <alternative>groupe_par</alternative>
25.      < /repli>
26.      <xsl :copy-of select="region[@id='video'] | region[@id='description']"/>
27.     < /seq_spatial>
28.     <seq_spatial id="seqSp2" alternative="yes">
29.      <équilibrage>center_W_father</équilibrage>
30.      <équilibrage>center_H_father</équilibrage>
31.      <xsl :copy-of select="region[@id='video'] | region[@id='description']"/>
32.     < /seq_spatial>
33.   < /seq_spatial>
34.   < /xsl :template>
35.  <xsl :template match="body">
36. ....
```

FIG. 5.11 – Exemple de couplage par transformation vers un langage cible

langage XEF pour supporter les éléments du langage SMIL et par conséquent il faut étendre le formateur XEF pour qu'il puisse formater ces éléments. Dans ce cas le langage XEF devient un langage de présentation à part entière.

Le processus d'extension du XEF présente certains inconvénients, notamment :

- Le langage XEF deviendrait un langage de présentation autonome. En effet, l'extension du XEF pour supporter les éléments de chaque langage source rendrait XEF équivalent à un langage de présentation universel. Or ce n'est pas l'objectif initial de XEF puisqu'il a été conçu pour offrir des services de formatage précis (gestion des priorités, des replis, ...) (cf.§.4.3) applicables sur des objets abstraits (temporels, spatiaux, ...),
- L'extension du formateur XEF va à l'inverse de notre ambition initiale de profiter des formateurs des langages de présentation actuels. En plus, le formateur XEF deviendrait très complexe et très lourd à gérer. En effet, la création d'un formateur dédié à un langage de présentation nécessite déjà un code conséquent (cf.§.chapitre 1), la construction d'un formateur qui gère plusieurs langages deviendrait très vite un processus très complexe.

Nous présentons dans le paragraphe suivant une autre approche basée sur la transformation qui tient compte du formateur du document source à contrôler.

5.5.3 Couplage par transformation vers un document composite

5.5.3.1 Présentation

L'étude de la première solution nous a montré qu'il n'est pas judicieux d'envisager un couplage basé exclusivement sur la décoration. En effet, l'expressivité des langages de sélection est assez limitée du fait qu'ils ne nous permettent pas de modifier le document source. Cependant, nous avons vu dans la seconde technique qu'il ne faut pas non plus envisager des solutions basées sur la génération d'un document spécifié dans un langage cible. En effet, dans cette technique il faudrait étendre le langage XEF pour chaque langage source.

Nous proposons dans cette troisième technique de prendre en compte le fait qu'un formateur du langage source existe et qu'on peut en profiter.

Dans cette approche, nous proposons une architecture basée sur la notion de document composite. Nous avons exploité la technique des *espaces de nommage* pour mettre en œuvre cette technique.

L'idée est de faire cohabiter des éléments des langages différents dans un seul document. L'auteur doit donc spécifier dans le document composite en plus du scénario spatio-temporel (via un langage de présentation de type : SMIL, SVG, Madeus,...), le contrôle de formatage qu'il souhaite (en utilisant le langage XEF). Le document composite est directement formaté par un formateur spécial, composé du formateur du langage source et du formateur XEF. Le formateur composite identifie clairement les éléments des deux langages par les espaces de nommage (cf.FIG.5.12).

Dans cette technique, l'auteur doit rééditer les documents existants pour rajouter des contrôles XEF. Une tâche qui n'est pas toujours facile. Il nous semble

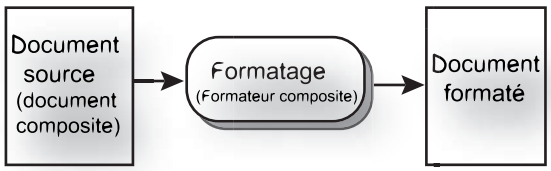

FIG. 5.12 – Processus de couplage en utilisant les espaces de nommage

intéressant de permettre à l'auteur de spécifier les contrôles qu'il souhaite avoir sans être contraint de rééditer le document source. Nous proposons d'améliorer l'architecture présentée dans la figure FIG.5.12 pour dépasser cette limite.

La solution que nous proposons consiste à ajouter une étape de transformation à l'architecture précédente. La figure FIG.5.13 illustre la nouvelle architecture de notre système. Le document source contient le scénario spatio-temporel de la présentation. La feuille de transformation contient les instructions de transformation et le document de contrôle comporte les éléments de contrôle XEF souhaités.

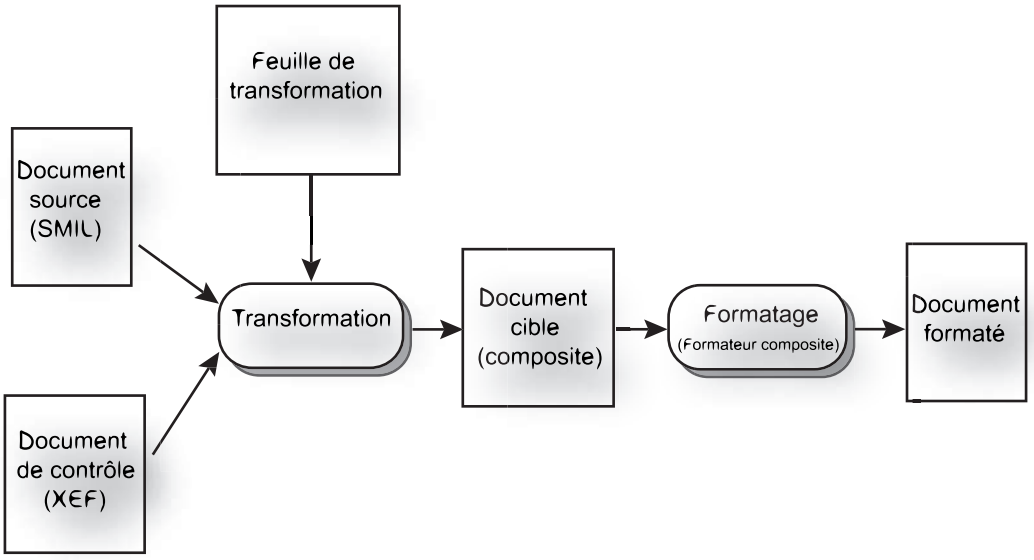

FIG. 5.13 – Couplage par transformation vers un document composite

A la différence de la technique de transformation présentée dans la section précédente où le document cible est spécifié dans un langage unique, le résultat de la transformation est un document composite. L'intérêt par rapport à la

technique de transformation vers un langage cible provient de la possibilité d'exploiter les formateurs des deux langages sans avoir besoin d'étendre aucun des langages.

5.5.3.2 Exemple

Nous donnons dans la figure FIG.5.14 un exemple d'un document composite permettant, en utilisant la technique des espaces de nommage, le couplage entre le document SMIL donné dans la figure FIG.5.1 et des éléments de contrôle du langage XEF.

Dans ce document, les déclarations *xmlns* sont placées dans l'élément racine du document. Nous avons défini deux espaces de nommage, le premier pour le langage SMIL et qui est déclaré par défaut ; le deuxième espace de nommage permet d'identifier les éléments du langage XEF. Le préfixe *xef* permet de distinguer entre les éléments des deux langages.

---

1. <smil xmlns="http//www.w3c.org/2001/SMIL20/Language" xmlns : xef=-"http//wam.inrialpes.fr/2003/xef10/Language" >
2. ...
3. <xef :seq_spatial id="seqSp0">
4. <xef :équilibrage>equilibrate_V_father</xef :équilibrage>
5. <smil :region id="town" top="15" left="200" height="45" width-="370"/>
6. <smil :region id="title" top="392" left="200" height="35" width-="370"/>
7. <xef :seq_spatial id="seqSp1">
8. <équilibrage>equilibrate_H_father</équilibrage>
9. <repli>
10. <relation>seqSp2</relation>
11. <alternate>groupe_par</alternate>
12. < /repli>
13. <smil :region id="video" top="80" left="20" height="292" width-="350"/>
14. <smil :region id="description" top="80" left="390" height="292" width="360"/>
15. < /xef :seq_spatial>
16. <xef :seq_spatial id="seqSp2" alternative="yes">
17. <xef :équilibrage>center_W_father</xef :équilibrage>
18. <xef :équilibrage>center_H_father</xef :équilibrage>
19. <smil :region id="video" top="80" left="20" height="292" width-="350"/>
20. <smil :region id="description" top="80" left="390" height="292" width="360"/>

```
21.      < /xef :seq_spatial>
22.     < /xef :seq_spatial>
23.    < /layout>
24.   < /head>
25.   <body>
26.    <seq xef :duration="3mn :50s" xef :reply="reduire(p1), alt(p3), suppression (p2)">
27.    <xef :groupe id="groupe_par">
28.     <par id="ParCinéma1" >
29.      <text id ="NomCinéma" src="Cinéma1.rtf" region="cinema"/>
30.      <xef :par_temporal id="RésuméFilm1">
31.       <xef :repli>
32.        <xef :relation>seq_R''esuméFilm1</xef :relation>
33.       < /xef :repli>
34.       <smil :video src="Film1.mpeg" region="video"/>
35.       <smil :text src="Discription1.rtf" region="description">
36.      < /xef :par_temporal>
37.      <xef :seq_temporal id="seq_R''esuméFilm1" alternative="true">
38.       <smil :video src="Film1.mpeg" region="video"/>
39.       <smil :text src="Discription1.rtf" region="description">
40.      < /xef :seq_temporal>
41.      <xef :par_temporal id="RésuméActeur1">
42.       <xef :repli>
43.        <xef :relation>seq_RésuméActeur1</xef :relation>
44.       < /xef :repli>
45.       <smil :video src="Dicaprio1.mov" region="video"/>
46.       <smil :text src="Dicaprio1.rtf" region="description">
47.      < /xef :par_temporal>
48.      <xef :seq_temporal id="seq_R''esuméFilm1" alternative="true">
49.       <smil :video src="Dicaprio11.mpeg" region="video"/>
50.       <smil :text src="Dicaprio1.rtf" region="description">
51.      < /xef :seq_temporal>
52. ....
53.   < /body>
54. < /smil>
```

FIG. 5.14 – Exemple de document composite SMIL + XEF

5.5.3.3 Discussion

Cette technique permet de coupler les deux langages tout en profitant des formateurs des deux langages.

Dans cette solution la problématique est repoussée au niveau du formatage. Comment le document composite va-t-il être formaté ? En effet, contrairement à l'inclusion classique dans un document (de type objets externes dans HTML comme les images, les applets,...), l'interdépendance des deux langages est forte. Il ne s'agit pas seulement d'inclure une boîte noire (quelque chose d'indépendant du reste) auquel on donne simplement une zone spatiale par exemple. Comme on peut le voir dans les lignes 4-25 de l'exemple donné dans la figure FIG.5.14, les éléments SMIL contiennent des éléments XEF qui eux-mêmes contiennent des éléments SMIL.

Cet enchevêtrement avec des interdépendances fortes rendent la mise en œuvre de la communication entre les formateurs complexe et nous étudions la façon de résoudre ce problème dans le prochain chapitre.

5.5.4 Synthèse

Les techniques de transformation que se soit une transformation vers un document spécifié dans un langage cible ou une transformation vers un document composite peuvent répondre à nos besoins de couplage du langage XEF avec d'autres langages de présentation.

Cependant ces techniques ont un certain nombre d'inconvénients. D'une part, même si les langages de transformations sont de type déclaratif, le processus d'édition (authoring) des feuilles de transformation est complexe. D'autre part, les techniques de transformation ne garantissent pas la validité du document cible.

5.6 Couplage sur l'instance formatée du document source

5.6.1 Présentation

La technique de couplage présentée dans le paragraphe précédent nécessite un couplage fort entre les formateurs, la dernière solution que nous proposons pour le couplage du langage XEF avec un langage de formatage prévoit un couplage faible entre les formateurs.

La figure FIG.5.15 présente le principe de cette technique. Le document de contrôle contient les éléments XEF à appliquer sur le document source. Ces éléments identifient les éléments du document source sur lesquels ils s'appliquent en utilisant un langage de sélection (par exemple les expressions XPath).

Il n'est pas nécessaire que le formateur source calcule les valeurs effectives des propriétés. il est tout à fait possible que le processus de formatage se décline à la génération d'une structure interne du document source (le parsing).

Comme nous le montrons dans le chapitre 6, les contrôles XEF peuvent

entraîner des incohéreneces dans le documents. Une façon de faire face à ce problème est de réexécuter le formateur source. Nous traitons ce problème en détail dans le chapitre 6.

Il est important de noter que le langage de sélection est utilisé seulement pour identifier les nœuds du document source sur lesquels on veut rajouter du contrôle. Le couplage effectif entre les deux langages s'effectue au moment du formatage. En effet, le formateur XEF extrait, en utilisant les expressions de chemin, une instance formatée de chaque élément du document source - d'où le nom que nous avons donné à cette technique - et il applique dessus le contrôle associé.

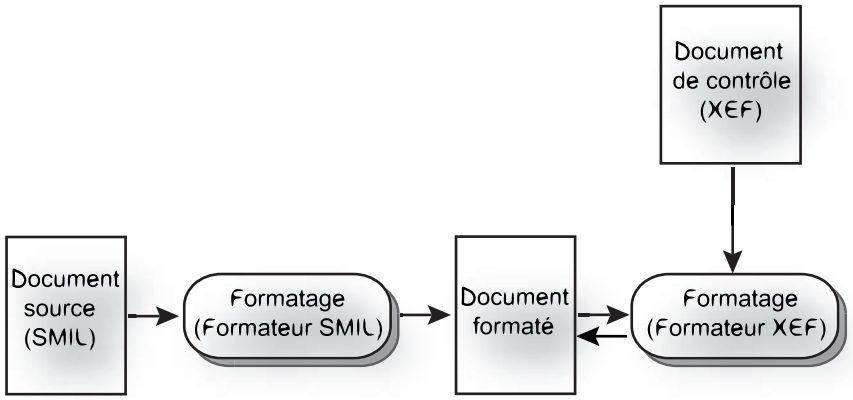

FIG. 5.15 – Couplage sur l'instance formatée du document source

5.6.2 Discussion

Cette technique permet de coupler les langages tout en minimisant l'interaction entre les formateurs. En effet, comme le montre la figure FIG.5.15, les deux processus de formatage sont bien séparés et la communication entre les deux formateurs s'effectue par l'intermédiaire du document source formaté.

Cependant, le couplage au moment du formatage pose deux problèmes liés à la sémantique du document source :

- D'une part, il faut identifier clairement les dépendances de certains éléments manipulés d'un côté ou de l'autre avec le reste du document pour pouvoir maintenir la cohérence du document,
- D'autre part, le document risque de ne plus respecter la spécification source. Ce point concerne aussi bien les *surcharges* amenées par XEF (exemple d'une séquence dans le document source formatée en un ensemble d'éléments joués en parallèle), que l'*effet de bord* qui risque d'entraîner des incohérences dans l'instance formatée du document. C'est notamment le cas des synchronisations événementielles, un exemple typique qui illustre ce probème est la suppression d'un élément dont la fin pro-

voque le démarrage d'un autre élément. C'est au moment de l'édition du document de contrôle qu'il faut régler ce genre de problèmes.

5.7 Synthèse

Nous avons donc présenté le couplage du langage XEF avec d'autres langages de présentation. Nous avons envisagé quatre possibilités pour ce couplage.

Le principe de la première technique proposée est d'utiliser un langage de sélection pour faire le couplage. Nous avons vu qu'en particulier ces langages ne permettent pas de modifier le document : un des besoins principaux du couplage du langage XEF avec d'autres langages de présentation.

Les techniques de transformation, quant à elles, surmontent ce problème en permettant l'ajout, la suppression et la modification des éléments du document. Dans une première variante, nous avons proposé de transformer le document source vers un autre document spécifié dans un langage de présentation supportant les deux langages d'entrée (le langage source et XEF). Pour ce faire, il faut passer par un processus d'extension du langage XEF. Cependant, cette démarche va à l'encontre de notre motivation initiale. Alors nous avons étudié une deuxième variante qui consiste à coupler les deux langages dans un seul document en utilisant le mécanisme d'espaces de nommage. Cette technique est conceptuellement intéressante : elle permet de répondre à nos besoins de couplage. Cependant, le processus de couplage des formateurs sous-jacents est complexe.

Enfin, nous avons proposé une architecture qui permet le couplage des langages en prenant en compte le processus de couplage des formateurs sous-jacents. Le principe est de faire le couplage au moment de formatage. Dans cette approche, la problèmatique est poussée au niveau des formateurs. Il s'agit principalement de définir le processus de coopération entre les deux formateurs pour éviter les incohérences qui peuvent surgir.

Parmi les techniques citées ci-dessus, il apparaît que seules celles basées sur la transformation vers un document composite et celle de couplage sur l'instance formatée du document répondent à nos besoins, mais le choix doit finalement se faire en fonction de l'architecture de couplage des formateurs.

Nous résumons dans le tabeau TAB.5.1 les caractéristiques de chaque technique proposée.

Technique de couplage	**Caractéristique**
Décoration	– utilise un langage de sélection, – Ne permet pas la modification de l'arbre du document
Transformation vers un langage cible	– Le couplage s'effectue en transformant les élements des deux langages vers un nouveau document décrit dans un seul langage, – Le langage cible doit supporter les éléments des deux langages, – Problème : il n'existe pas un langage de présentation qui supporte le langage XEF
Transformation vers un document composite	– Le couplage s'effectue en transformant les élements des deux langages vers un document composite, – Utilise le mécanisme des espaces de nommage, – Processus de formatage très complexe.
Couplage sur l'instance formatée	– Utilise un langage de sélection, – Couplage au moment du formatage, – Problème : comment gérer la coopération entre les deux formateurs.

TAB. 5.1 – Tableau récapitulatif des techniques de couplage proposées

Chapitre 6

Une architecture logicielle pour le couplage du formateur XEF avec un formateur existant

6.1 Introduction

Dans ce chapitre, nous décrivons l'architecture logicielle que nous proposons pour le couplage des formateurs des langages de présentation.

Nous identifions dans un premier temps les besoins propres au couplage du formateur XEF avec d'autres formateurs. Ensuite, nous donnons quelques détails techniques sur le processus de formatage et enfin, nous étudions deux approches pour le couplage des formateurs.

6.2 Identification des besoins

Nous voulons réaliser un formateur composite à partir de formateurs existants. Le principal challenge pour la conception/réalisation d'une architecture pour la composition des formateurs est la gestion de *la communication effective* entre les formateurs à coupler. Par communication effective, nous entendons :

- Le couplage entre les formateurs doit être aussi faible que possible, afin que le système de couplage soit réutilisable par plusieurs formateurs,
- Le formateur composite doit assurer la *cohérence* entre les différents formateurs. En effet, les traitements effectués par un formateur peuvent remettre en cause les valeurs de propriétés déjà calculées par l'autre formateur ou les valeurs des propriétés d'éléments ayant des liens entre eux. Ce formateur composite doit en tenir compte et proposer des solutions. Comme nous allons voir dans le paragraphe (§.6.6), ce problème est dû

principalement aux *dépendances* qui peuvent exister entre les éléments. En effet, les langages de présentation permettent de spécifier des scénarios spatio-temporels dont les éléments ne sont pas indépendants. Le formateur composite doit assurer la propagation des traitements effectués par un formateur sur un élément à l'ensemble des éléments dépendants,
- Enfin, un dernier besoin qui découle des deux précédents est que chaque formateur doit pouvoir accéder à toutes les données dont il a besoin pour mener à bien le processus de formatage. En contre partie, il doit fournir toutes les informations dont les autres formateurs ont besoin. Par exemple, dans le contexte de contrôle XEF du formatage d'un document SMIL, le formateur XEF doit pouvoir accéder aux attributs d'un élément *region* pour équilibrer spatialement plusieurs régions.

L'influence du langage de présentation et donc de l'architecture du formateur associé doit rester aussi faible que possible dans le choix de l'architecture du système de couplage pour que ce dernier soit réutilisable pour différents langages de présentation.

6.3 Choix d'une architecture logicielle

Le choix de l'architecture logicielle de couplage est aussi important que celui du couplage des langages. Notons que ces choix ne sont pas totalement indépendants l'un de l'autre. Ils s'interpénêtrent et leurs solutions ne peuvent être envisagées de manière séparée.

L'état de l'art dressé dans le chapitre 3, nous a permis de nous orienter vers une architecture logicielle à composant.

Il est communément admis que la description d'une architecture dans le paradigme de la programmation orientée composant comprend au moins la définition : de *composants*, de *connecteurs* et d'une *configuration*.
- *Les composants* : ce sont des entités de traitement de données. Dorénavant, les formateurs sont considérés comme des composants dont la fonction est de formater les éléments du langage associé,
- *Les connecteurs* : ce sont des entités complémentaires [Sha95] aux composants, ils spécifient la sémantique de la communication permettant à ces composants de travailler ensemble. La description d'un connecteur comprend, d'une part, la définition des composants intéressés par la communication, et d'autre part, celle du protocole spécifiant les interactions entre ces composants,
- *La configuration* : cette propriété consiste à adapter le comportement d'un composant à un contexte d'utilisation particulier. Par exemple, adapter le formateur XEF pour qu'il puisse traiter des éléments spécifiques à un langage de présentation.

6.4 Choix du type de composition

Cette section reflète l'état actuel des travaux s'intéressant à la composition des architectures à composant. Le but est d'identifier le type de composition qui

répond le mieux aux besoins de composition des formateurs cités ci-dessus.

Une vaste bibliographie de recherche peut être trouvée aujourd'hui quant aux techniques de composition des architectures logicielles. Une synthèse de ces travaux a été faite dans [CLBBD01, Duc02].

Parmi les techniques de composition actuelles, nous trouvons :

- *Composition structurelle* : elle fournit le plus bas niveau d'abstraction dont les autres types de composition ont besoin. Elle présente deux mécanismes complémentaires :
  - La gestion des *liaisons* (dépendances) entre composants, exprimées en termes de types d'interfaces : les services fournis et les services requis,
  - La décomposition hiérarchique, en termes de composants et sous-composants. Contrairement au premier mécanisme, ce dernier n'est pas réellement indispensable,
- *Composition fonctionnelle* : elle permet de séparer et d'assembler des *vues fonctionnelles* pour obtenir une seule vue globale du système. Le comportement du système composé est une combinaison du comportement de chaque vue. Les techniques les plus représentatives de ce type de composition sont : la programmation par aspects (AOP)[KLMMLLI97], la programmation par sujet (SOP) [OHBS94] et les filtres de composition [BAT01].
- *Composition contractuelle* : elle consiste à vérifier que la combinaison de plusieurs *contrats* forme un contrat valide plus global. Un contrat est une spécification permettant de relier certaines obligations d'un composant à certaines hypothèses sur son environnement. Elle définit soit des obligations, soit des hypothèses qui ne sont pas obligatoires.

Dans notre contexte, nous envisageons de créer un formateur par composition des formateurs existants avec un couplage faible. Nous avons choisi de mettre en œuvre une composition structurelle. En effet, nous voudrions obtenir un nouveau formateur dont la structure s'appuie sur la composition des formateurs existants. Les interactions doivent être spécifiées séparément des formateurs sur lesquels elles s'appliquent et donc que les interactions devraient être définies en tant qu'entité propre.

6.5 Rappel sur le formatage

Nous avons introduit le processus de présentation d'un document multimédia au chapitre 1. Celui-ci consiste en deux étapes : la transformation et le formatage. Nous donnons dans le présent paragraphe quelques détails techniques supplémentaires qui nous seront utiles lors de la spécification du système de couplage.

Le processus de formatage comprend deux étapes :

- *Etape d'analyse* : cette étape a pour objectif de transformer le document source en une structure de données interne appelée *arbre de formatage* (formatting tree). Ensuite, l'arbre de formatage obtenu est raffiné par itérations successives. En effet, certaines éléments vont hériter de propriétés d'autres éléments, certains attributs sont définis relativement à

d'autres attributs : il faut donc expliciter toutes ces relations et calculer les valeurs de chacun des attributs de chacun de éléments.
- *Etape de formatage* : la deuxième étape est la construction de l'*arbre des aires* (area tree). Il s'agit de définir la disposition des éléments sur une ou plusieurs pages, comme spécifié dans le document. Cette phase sera évidemment différente suivant le support vers lequel on souhaite exporter le document.

Plusieurs techniques ont été proposées pour représenter l'arbre de formatage :
- *DOM* : le document source est représenté par une structure hiérarchique d'objets en mémoire, l'arbre de formatage est créé par accès à la structure du document et son contenu par l'intermédiaire des APIs DOM,
- *SAX* : le document source est analysé en un flux d'évènements SAX. L'arbre de formatage est créé par l'application qui manipule l'API SAX,
- *Techniques ad hoc* : dans les deux premières techniques la création de l'arbre de formatage passe par une représentation intermédiaire (DOM ou SAX), il existe également des applications qui permettent de créer cet arbre directement sans passer par une représentation intermédiaire. Ces applications font appel à des techniques ad hoc.

La technique utilisant DOM permet un accès aléatoire à tous les constituants du document. Cette technique est utilisée chaque fois que le traitement nécessite de faire des modifications dans le document source.

Dans le cas des techniques basées sur SAX et des techniques ad hoc il n'existe pas une représentation tangible du document source. Cela ne signifie pas pour autant que l'application ne manipule pas une structure interne représentant ce document.

Dans le cadre du langage XEF, nous serons toujours amenés à modifier, voir à ajouter et à supprimer des nœuds (élément, attribut ou texte). Ceci veut dire que nous devons avoir une représentation complète de la structure et du contenu du document source. Pour satisfaire ce besoin, nous sommes soumis à l'obligation de manipuler une représentation du document source sous forme d'un arbre DOM.

6.6 Problèmatiques de la composition des formateurs

L'objectif de cette section est de mettre en évidence les principales problèmatiques de la composition des formateurs des langages de présentation. En effet, indépendamment du choix du type de composition, nous avons identifié un certain nombre de problèmes à résoudre.

Nous proposons dans cette section de faire un point sur ces différents problèmes. Il ne s'agit pas d'évaluer, cas par cas, la composition du formateur XEF avec chaque formateur de langage source, mais de faire un bilan des problèmes communs que peuvent poser la composition des formateurs. Nous n'abordons pas ici les problèmes de conception de l'architecture, comme par exemple la spécification des protocoles de communication entre les formateurs. Mais, nous nous intéressons exclusivement aux problèmes qui doivent être réglés dans la

spécification de l'architecture de composition, à savoir :

- Les dépendances entre les éléments du document source,
- Les incohérences que peuvent entraîner les traitements du formateur XEF,
- Le problème des propriétés non calculées,
- Le mode de coopération entre les formateurs.

6.6.1 Dépendances entre les éléments

Comme nous le mentionnions au chapitre 1, nous nous situons dans un contexte où la notion de *scénario* est primordiale. La définition d'un scénario consiste à spécifier toutes les relations d'ordonnancement des éléments du document que se soit des relations temporelles ou spatiales.

De nombreux travaux, dont ceux de [All91, KL91, Lay97] pour le temporel, ont été effectués sur l'expression des relations spatio-temporelles dans un scénario. La maturité de ces travaux est la spécification d'un ensemble de relations qui peuvent relier les objets entre eux. La majorité des langages de présentation actuels permet à l'auteur de définir de telles relations (cf.chapitre 1).

Ces relations peuvent être classées en deux catégories :

- *Relations simples* : ce type de relation permet d'exprimer des liens qualifiés de *base* entre les éléments. Elles sont de deux types :
  - Des relations qui n'impliquent pas de paramètres numériques,
  - Des relations qui impliquent des paramètres numériques. Dans le cas où ces paramètres ne sont pas spécifiés, ils sont définis par défaut.
- *Relations causales* : ces relations expriment les liens de causalité (événementiels) entre les éléments. Dans l'exemple de la figure FIG.6.1, la fin de l'élément B provoque la fin de l'élément *par*, qui lui même provoque la fin de l'élément C.

Ces dépendances nécessitent généralement le recours à des traitements récursifs. En effet, les traitements que peuvent subir un élément ayant des liens avec d'autres éléments peuvent très bien remettre en question certaines propriétés de ces éléments. Les deux formateurs doivent garder trace de l'ensemble de ces liens pour pouvoir assurer les traitements inhérents à ces dépendances.

Pour illustrer ce problème, considérons une autre fois l'exemple donné dans la figure FIG.6.1. Un formateur XEF, en appliquant un contrôle *flexible(B)*, risque de remettre en cause les durées des éléments *par*, C et A et peut être d'autres éléments extérieurs à l'élément *par*. Il faut donc un mécanisme permettant de garantir la propagation des traitements entre les éléments liés.

Plus généralement, l'application d'un contrôle XEF sur un scénario revient à le modifier et donc ce traitement est similaire à l'effet d'une opération d'édition.

Le système de couplage des formateurs doit être doté d'un mécanisme permettant de maintenir les relations qui peuvent exister entre les éléments.

En particulier, le traitement des éléments dépendants d'autres éléments peut engendrer des *incohérences* dans le scénario.

```
1. <par endsync="B">
2.   <video id="A" src="fofo.mpeg" begin="B.begin+2s">
3.   <audio id="B" src="toto.au">
4.   <text id="c" src="fofo.mpeg">
5. </par>
```

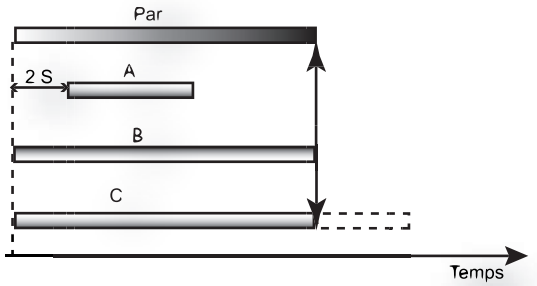

FIG. 6.1 – Relations causales

6.6.2 Incohérences

Dans une optique voisine de la première problèmatique ci-dessus, le traitement d'un document par plusieurs formateurs pose un véritable problème de cohérence du scénario. En effet, l'ajout d'un contrôle XEF peut conduire à un ajout ou à une suppression d'éléments ou de relations entre ces éléments. Ces modifications peuvent introduire des incohérences dans la spécification du scénario du fait que ces relations et ces éléments ne sont pas indépendants (cf.§. ci-dessus).

Quelques travaux ont déjà traité ce problème (cf.[Lay97]) mais dans le cadre d'un seul formateur. Nous proposons d'étudier ces problèmes dans le cas des formateurs composites. Plusieurs types d'incohérence peuvent émerger lors du formatage : incohérences causales, incohérences qualitatives et incohérences quantitatives.

6.6.2.1 Incohérences causales

Ce type d'incohérences est dû aux relations causales qui peuvent exister entre les éléments et les relations du document source.

Supposons qu'une partie d'un document SMIL (cf.FIG.6.2) est composée de trois éléments A, B et C, et que ces éléments sont liés par des synchronisations sur les attributs *begin* et *end* comme le montre la figure FIG.6.2.A. Supposons aussi que le document de contrôle XEF spécifie un contrôle de type *suppression*

sur l'élément B. A un moment donné le formateur XEF exécute ce contrôle et supprime l'élément B. Le scénario devient incohérent du fait que les deux autres éléments ont des débuts qui ne peuvent jamais être résolus (cf.FIG.6.2.B). Dans ce cas, la spécification SMIL [SMIL2] propose pour résoudre ce problème d'ignorer simplement l'attribut *begin* des éléments A et C. Leur date de début sera celle de leur parent. Mais est-ce bien ce que voulait l'auteur ?

1. <video id="A" src="fofo.mpeg" begin="B.begin+2s">
2. <audio id="B" src="toto.au" >
3. <text id="c" src="fofo.mpeg" begin="B.end+1s">

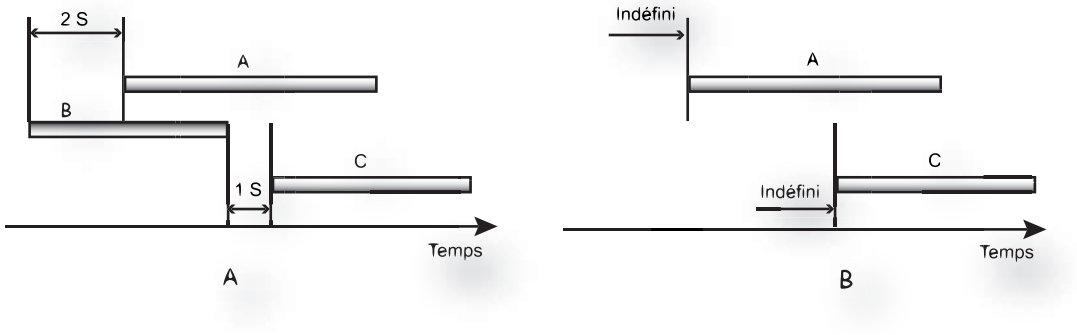

FIG. 6.2 – Incohérence causale

6.6.2.2 Incohérences qualitatives

Les incohérences qualitatives correspondent à l'introduction dans le scénario des modifications qui affectent la sémantique des éléments.

Pour illustrer ce problème, [Lay97] donne un exemple intéressant. Supposons qu'une partie d'un document Madeus est composée de trois éléments A, B et C, et que ces éléments sont liés par les relations A *meets* B et B *meets* C. Un contrôle XEF qui ajoute une nouvelle spécification de type C *overlaps* A rend le scénario incohérent (cf.FIG.6.3). Ce type d'incohérence ne dépend pas des durées des éléments mais de la sémantique même des relations mises en jeu.

6.6.2.3 Incohérences quantitatives

L'ajout d'un contrôle XEF peut laisser le scénario cohérent sémantiquement mais pas quantitativement. Nous illustrons ce cas de figure par l'exemple présenté dans la figure FIG.6.4. Dans un document Madeus, l'auteur a spécifié les relations A *starts* B et B *finishes* A (cf.FIG.6.4.A). Si après l'échec du formateur Madeus, le formateur XEF exécute un contrôle *réduction (B)*. Les deux éléments

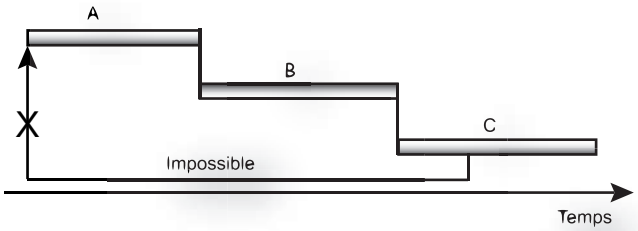

FIG. 6.3 – Incohérence qualitative

ne peuvent terminer en même temps comme le suggère la relation *B finishes A* (cf.FIG.6.4.B). En effet, la durée de *B* devient trop petite pour pouvoir assurer à la fin les deux relations précécentes.

Bien qu'il soit cohérent qualitativement, le scénario est incohérent quantitativement.

FIG. 6.4 – Incohérence quantitative

6.6.2.4 Synthèse sur les incohérences

Dans cette section nous avons présenté l'un des problèmes majeurs auxquels nous sommes confrontés lors de la composition des formateurs. Nous ne sommes bien sûr pas entrés dans les détails de chaque type d'incohérences puisqu'elles sont équivalentes à celles rencontrées dans les applications d'édition. Enfin, même si les exemples donnés ci-dessus pour illustrer ce problème ne

considèrent que l'aspect temporel, il est tout à fait possible de rencontrer des incohérences similaires entre les relations spatiales.

Quelque soit le type d'incohérence, il est clair que ce problème constitue l'un des obstacles majeurs à surmonter dans la composition des formateurs. *Le formateur composite doit donc proposer des mécanismes permettant d'assurer la cohérence du scénario.*

6.6.3 Propriétés non-calculées

Dans le cas idéal, le système de formatage calcule les valeurs de toutes les propriétés avant l'exécution du document. Il peut, cependant, ne pas avoir calculé toutes ces valeurs lors du formatage. On se trouve alors dans l'un des deux cas suivants :

- Ces propriétés correspondent à des enchaînements qui sont définis en fonction de données connues uniquement au moment de la présentation (par exemple, occurrence d'une interaction utilisateur). On parle alors de *scénario indéterministe* [Lay97],
- Les valeurs de ces propriétés dépendent d'autres propriétés et le système de formatage - même si cela serait tout à fait possible - ne calcule pas cette dépendance (c'est-à-dire, n'attribue pas à la propriété sa valeur effective). Les valeurs effectives sont calculées au moment de l'exécution ce qui est le cas du formateur SMIL de l'environnement XSmiles [XSmiles].

La composition des formateurs - en particulier du formateur XEF avec d'autres formateurs - s'apparente au calcul des valeurs des attributs. En effet, XEF n'est pas un langage de formatage indépendant mais est un service de contrôle de formatage qui est appliqué sur des propriétés dont la valeur est *contrôlable*. Une propriété a une valeur contrôlable si un intervalle de valeur lui est associé.

Ce problème n'est pas très génant. En effet, on peut appliquer un contrôle XEF sur un ensemble d'éléments même si les valeurs de leurs propriétés ne sont pas calculés.

6.6.4 Dépendances entre les formateurs

Si les problèmes de dépendance et de cohérence ont été largement étudiés dans le domaine de l'édition de documents électroniques, il n'en reste pas moins que le processus de coopération entre les formateurs n'a jamais été abordé mis à part les travaux qui portent sur la coopération de résolveurs de contraintes, les plugins et les documents composites. En effet, l'utilisation de plugins et de documents composites [DFK01] a commencé à initialiser de tels besoins, mais les travaux sont pour l'instant restés sur des approches minimisant (voir interdisant) les dépendances entre les formateurs (principe de la boite noire). Des coopérations intéressantes ont également été proposées du coté des résolveurs de contraintes, mais ces travaux ne sont pas transposable à notre approche qui est de plus haut niveau.

Le choix du mode de coopération aura bien évidemment un impact important sur la stratégie à mettre en place pour résoudre les problèmes iden-

tifiés précédemment. Nous avons identifié deux techniques possibles pour faire coopérer les formateurs : coopération en *parallèle* et coopération en *série*.

Nous montrons dans la section (§.6.7) que dans la coopération en parallèle nous sommes confrontés à un problème d'ordonnancement NP-difficile. Alors nous proposons une architecture de composition en série qui permet d'éviter ce problème.

Le mode de coopération entre les formateurs à composer révèle un autre problème important, celui de l'ordre d'exécution des formateurs. Pour illustrer ce problème, considérons un élément *par* du langage SMIL ayant une durée de 10s et contenant deux éléments : une *vidéo* et un élément *seq* qui lui même se compose de quatre éléments (des média ou des éléments composites) : A, B, C et D. Après un équilibrage XEF les éléments A, B, C et D ont les durées 5s, 3s, 2s et 4s respectivement (cf.FIG.6.5.A). Soit un formatage SMIL qui entraîne la situation présentée dans la figure FIG.6.5.B (l'élément D ne se joue pas). Il faut donc exécuter le formateur XEF une autre fois pour équilibrer ces éléments à nouveau (cf.FIG.6.5.C).

Dans ce cas particulier, si le formateur XEF n'intervient que localement, il aurait être plus éfficace de commencer le formatage par l'exécution du formateur SMIL et ensuite le formateur XEF. Par contre, si le formateur XEF intervient à plusieurs endroits différents dans l'arbre du document, on a un problème NP-difficile àrésoudre.

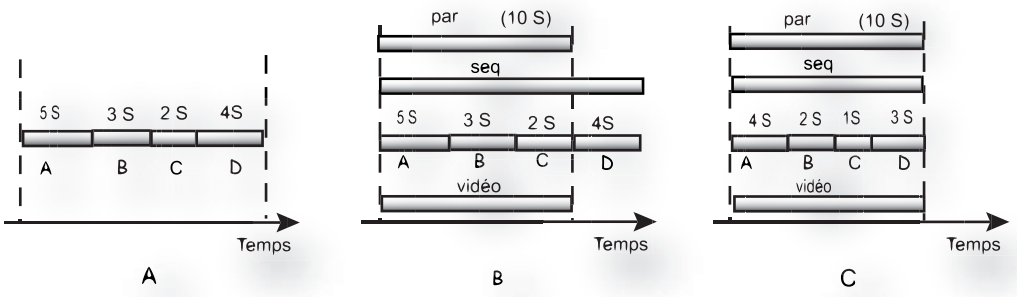

Fig. 6.5 – Effet de l'orde d'exécution des formateurs

6.7 Quelle architecture pour le couplage des formateurs ?

Dans les sections précédentes, nous avons fixé le cadre de l'architecture logicielle et le type de composition souhaité, ainsi que les problèmes qui y sont associés. Nous proposons maintenant d'étudier en détail deux modes de composition des formateurs : la composition en parallèle et la composition en série.

Il semble naturel d'opter pour une architecture de composition en parallèle, c'est-à-dire, un formateur composite qui permet le formatage des éléments en respectant leur ordre de couplage au niveau des langages mais nous montrons que ce n'est pas la meilleure solution. Nous proposons en effet une solution améliorée avec plusieurs variantes.

Notations

Notons F_{source} le formateur du document source à contrôler et P_{source} l'ensemble des propriétés des éléments du langage source. Nous notons le processus de formatage[1] des propriétés P_{source} par :

$$F_{source} \models P_{source}$$

$$\text{où : } P_{source} = \{P_i \mid P_i \textit{ peut être formatée par } F_{source}\}$$

de même pour le formateur XEF :

$$F_{XEF} \models P_{XEF}$$

$$\text{où : } P_{XEF} = \{P_i \mid P_i \textit{ peut être formatée par } F_{XEF}\}$$

Le formateur composite F (noté $F = F_{source} \oplus F_{XEF}$) permet de formater les propriétés P tel que :

$$P = \{P_i \mid P_i \subset P_{source} \vee P_i \subset P_{XEF}\}$$

Typiquement, si le formateur source est un formateur MathML, alors :

$$P_{source} = P_{MathML} = \{lambda, apply, ci, plus, \ldots\};$$

$$P = \{lambda, apply, ci, plus, \ldots, repli, priorité, \ldots\}$$

6.7.1 Définitions

Un *comportement* est un ensemble d'opérations avec un ensemble de contraintes qui déterminent quand elles peuvent intervenir.

Nous distinguons deux types d'actions (par exemple : des appels) dans la composition des formateurs : des actions *locales* et des actions *globales*.

- *Les actions locales* : désignent l'ensemble des comportements propres à un formateur. Autrement dit, l'ensemble des opérations que peut effectuer un formateur indépendamment des autres formateurs. L'ordonnancement de ces actions est à la charge du formateur en question,
- *Les actions globales* : désignent l'ensemble des comportements induits par les appels entre les deux formateurs. Ces actions ne sont pas ordonnées entre elles. Des contraintes peuvent s'y appliquer pour les synchroniser. Par la suite, nous ne nous intéressons qu'aux actions globales.

[1] Rappel : formater une propriété consiste à lui assigner une valeur effective.

6.7.2 Composition en parallèle

Cette approche consiste à faire exécuter les deux formateurs en même temps. Le point de départ est le formateur source. Plus formellement, la composition parallèle, dans un contexte de temps logique des comportements des composants (formateurs) signifie que le modèle d'exécution (le formateur composite) a la possiblité de choisir l'ordre d'exécution des actions globales des composants (formateurs) dans un contexte de temps physique.

$$formatage_{parallèle} \Leftrightarrow \{formatage_{source}^{P_k} \mid formatage_{XEF}^{P_{k'}}\}^*$$

- $formatage_{source}^{P_k}$ désigne le processus de formatage source de la propriété P_k.
- Le symbole « * »spécifie que l'ensemble $\{Formatage_{source}^{P_k} \mid Formatage_{XEF}^{P_{k'}}\}$ peut contenir zéro ou plusieurs appels,
- Le symbole « | »indique que l'ordre n'est pas important,

La communication entre les deux formateurs se définit à partir de leurs *séquences d'exécution*. Une séquence d'exécution se traduit par une suite de traitements élémentaires (par exemple, le formatage d'un élément).

La composition des formateurs génère un ensemble *d'entrelacements* des séquences d'exécution possibles de ces formateurs.

Deux séquences d'exécution appartenant chacune à un formateur peuvent être *conflictuelles*. Par exemple, il est fréquent qu'une séquence d'exécution d'un formateur $F1$ modifie une propriété déjà calculée par un autre formateur $F2$. C'est le formateur composite qui doit gérer la communication, et donc la cohérence du scénario, entre les deux formateurs.

Dans la figure FIG.6.6, nous avons considéré qu'il y a un seul document composite contenant les éléments des deux langages. Il est tout à fait possible que ces éléments se trouvent dans deux documents séparés et le lien s'effectue entre les deux documents en utilisant la technique de couplage des langages présentée dans le paragraphe (§.5.6). Dans ce cas, le formateur composite F doit pouvoir gérer les liens entre les deux documents (cf.FIG.6.7).

Le mécanisme de composition mise en œuvre dans ces architectures est celui de l'inclusion. En effet, les interfaces des formateurs source et XEF sont totalement cachés par le formateur composite. La communication avec les objets externes (par exemple le document) s'effectue à travers les interfaces du formateurs composite.

Nous détaillons plus formellement le principe de cette approche dans le paragraphe suivant.

6.7.2.1 Principe

Dans ce type de composition, la problèmatique est d'arriver à gérer les appels entre les deux formateurs.

Nous appelons *appel orienté*, un couple $({F_i}^{P_K}, F_j)$ tel qu'il soit spécifié dans la description de l'architecture de couplage que les deux formateurs commu-

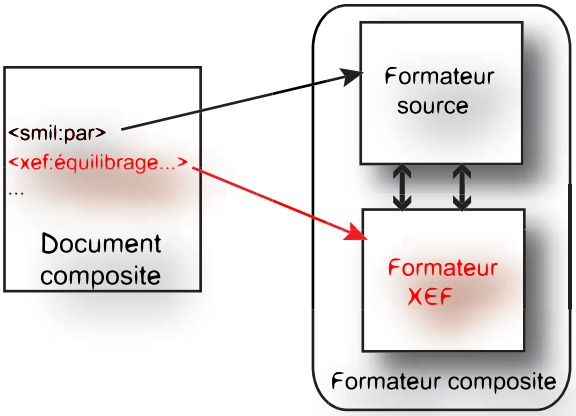

FIG. 6.6 – Composition en parallèle des formateurs à partir d'un document composite

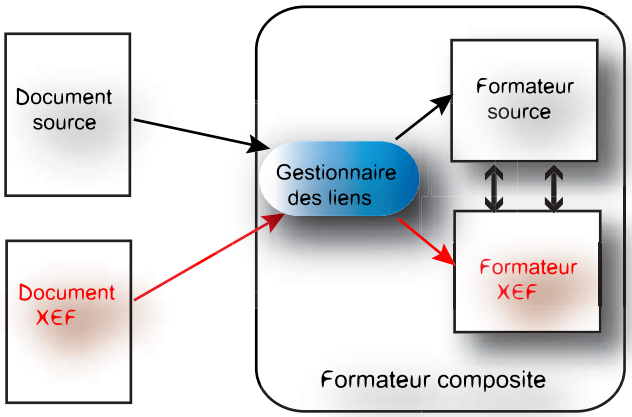

FIG. 6.7 – Composition en parallèle des formateurs à partir de documents indépendants

niquent dans le sens $F_i \longrightarrow F_j$ et que le formateur F_i requiert un service de formatage fourni par le formateur F_j pour formater la propriété P_K.

$\forall i, j \in \{source, XEF\} \wedge (i \neq j), Appel\ ({F_i}^{P_K}, F_j) \Rightarrow (F_i \rightarrow F_j) \wedge (F_j\ formate\ P_K)$

- Nous appelons *émetteur* de l'appel orienté $({F_i}^{P_K}, F_j)$ le formateur F_i,
- Nous appelons *récepteur* de l'appel orienté $({F_i}^{P_K}, F_j)$ le formateur F_j.

Comme évoqué précédemment, le formateur récepteur doit connaître les dépendances entre les éléments et leurs propriétés. Soit $NS^{{P_k}^t}_{F_i \rightarrow F_j}$ l'ensemble

des propriétés dont le formateur récepteur a besoin pour formater la propriété P_k dans un appel $({F_i}^{P_K}, F_j)$ effectué à un instant t.

Après cet appel, comme nous l'avons vu précédemment, certaines propriétés seront remises en question. Soit $RS_{F_i \to F_j}^{{P_k}^t}$ l'ensemble de ces propriétés. Ces dernières peuvent être des propriétés XEF ou des propriétés du langage source.

$$RS_{F_i \to F_j}^{{P_k}^t} = \{P_H \mid (P_H \in P_{source}) \vee (P_H \in P_{XEF})\}$$

Il est possible que : $RS_{F_i \to F_j}^{{P_k}^t} \cap NS_{F_i \to F_j}^{{P_k}^t} \neq \emptyset$. Ce cas de figure pose le problème de cycle.

Pour assurer la cohérence du scénario, le formateur F doit recalculer les valeurs des propriétés $RS_{F_i \to F_j}^{{P_k}^t}$. Pour ce faire, nous distinguons deux solutions possibles :

Première possibilité

Le formateur composite F arrête le formateur récepteur et fait des appels de type :

$$\begin{cases} Appel\ (F_{récepteur}^{P_H}, F_{émetteur}) \\ avec\ P_H \in (RS_{F_i \to F_j}^{{P_k}^t} \cap P_{émetteur}) \end{cases}$$

Pendant son traitement, le formateur émetteur pourrait tomber à nouveau sur des propriétés $P_G \in P_{récepteur}$ et donc effectuer des appels :

$$Appel\ (F_{émetteur}^{P_G}, F_{récepteur})$$

Deuxième possibilité

Le formateur récepteur continue le formatage des propriétés $P_{K'}$ telles que :

$$P_{K'} \in (RS_{F_i \to F_j}^{{P_k}^t} \cap P_{récepteur})$$

Après avoir terminé le formatage de ces propriétés, il effectue des appels à l'autre formateur pour qu'il traite ses propriétés.

D'une manière générale, la communication entre les deux formateurs peut être donnée par la formule suivante :

$$\forall i, j \in \{source, XEF\} \wedge (i \neq j),\ communication\ (F_i, F_j) \Leftrightarrow \{Appel\ (F_i, F_j); Appel\ (F_j, F_i)\}^*$$

Ce qui revient à résoudre un problème d'*ordonnancement de tâches*. En effet, la composition parallèle des formateurs doit s'appuyer sur l'expression des contraintes liées à l'entrelacement des séquences d'exécution, et donc sur la synchronisation des comportements des formateurs. Ces contraintes sont dites des *contraintes d'ordonnancement*.

6.7.2.2 Définition

Une *contrainte d'ordonnancement* spécifie une relation d'ordre temporel entre deux séquences d'exécution. Ces contraintes peuvent porter sur la concurrence, la séquentialité,...[ISO95].

Avant d'aborder le thème d'ordonnancement, nous donnons un exemple concret illustant cette technique.

6.7.2.3 Exemple

Dans l'exemple présenté dans la figure FIG.6.8, nous avons :

$$P_{source} = P_{SMIL} = \{video, par, seq\}$$

$$P_{XEF} = \{repli, réduction\}$$

Supposons qu'à l'instant t le formateur SMIL entre dans un cas d'échec, par exemple la durée de la présentation dépasse la durée spécifiée par l'auteur. Dans ce cas, le formateur composite F reformate le document en prenant en compte les éléments de contrôle XEF placés sur les éléments SMIL. Dans l'exemple, à la ligne 1 l'élément $smil : par$ est reformaté en appliquant dessus le contrôle $xef : repli = "réduire"$. Ceci est caractérisé par un appel (le symbole $\leadsto$ désigne la relation qui existe entre deux éléments) :

$$Appel\ (F^{repli}_{SMIL}, F_{XEF})$$

$$\text{avec : } RS^{repli^t}_{F_{SMIL} \to F_{XEF}} = \{(video1 \leadsto seq1)\}$$

Après cet appel, les valeurs (les durées) des éléments fils de *par* sont changées. Dans ce cas, les valeurs des propriétés de l'ensemble $RS^{repli^t}_{F_{SMIL} \to F_{XEF}}$ doivent être recalculées.

---

1. <smil :par id="RésuméFilm1" xef :repli="réduire">
2. <smil :video id="video1" src="Film1.mpeg" region="video"/>
3. <smil :video src="Description1.mpeg" region="description">
4. <smil :video src="Description2.mpeg" region="commentaire">
5. < /smil :par>
6. ...
7. <smil :seq id="seq1" begin="dur(video1)+2s">
8. ...

FIG. 6.8 – Exemple illustrant la composition parallèle

6.7.2.4 Ordonnancement des tâches

Nous traitons dans le paragraphe suivant le problème d'ordonnancement des tâches entre les actions globales. Nous ne traitons pas le problème d'ordonnancement des tâches locales. Ce dernier est censé être traité localement par chaque formateur.

Dans la terminologie de la recherche opérationnelle, un problème d'ordonnancement désigne tout problème dans lequel l'objectif est l'allocation de ressources au cours du temps, de façon à réaliser un ensemble de tâches. Des définitions plus précises raffinant cette définition très générale peuvent être trouvées dans [Kan76, CC88, Pin95].

De nombreux travaux ont permis de distinguer plusieurs variantes du problème d'ordonnancement selon la nature des opérations à réaliser (morcelables ou non, répétitives), les caractéristiques des ressources (consommables ou renouvelables, interchangeables), les contraintes portant sur les opérations (dates de disponibilités, précédences) et les critéres à optimiser (retard ou coût total).

Dans notre contexte, nous sommes confronté à un problème d'*ordonnancement disjonctif*. Ce terme qualifie un problème dans lequel certaines tâches se partagent une ou plusieurs ressources qui ne peuvent être consacrées qu'à une seule opération à un instant donné.

La plupart des problèmes appartenant à cette catégorie sont classés *NP-difficiles* au sens fort [GJ79], il n'existe donc pas d'algorithme polynomial connu pour les résoudre de façon exacte.

Généralement pour résoudre de tels problèmes, la plupart des solutions proposées s'appuient sur des techniques d'énumération implicite de l'espace des solutions pour obtenir ou se rapprocher de l'optimum d'un tel problème. Le facteur clé permettant de limiter l'explosion combinatoire inhérente à ce type de méthode réside incontestablement dans l'utilisation de *bornes inférieures* et *règles d'élimination de qualité*, les premières concourant à la limitation de l'arborescence de recherche et les secondes tentant de réduire l'espace des solutions à ses seules parties pertinentes.

Plusieurs techniques ont été proposées pour mettre en oeuvre le principe des règles d'élimination disjonctives[2]. Le principal dénominateur de ces techniques est de s'appuyer sur les fenêtres d'exécution des tâches pour tronquer l'espace des solutions. En général, ces règles spécifiques (baptisées *opérations locales*) procèdent soit par identification d'une relation d'ordre partielle entre opérations, soit par réduction directe des bornes des fenêtres associées aux tâches. Dans le premier cas, on parle d'*arbitrage*, identifiant par exemple que dans toute solution optimale telle opération figure toujours avant telle autre, tandis que la seconde catégorie de règle est qualifiée d'*ajustement* ou de *réduction de domaine* de fenêtre temporelle, l'objectif étant de réduire les fenêtres à leur plus simple expression. La dichotomie entre ces deux concepts n'est qu'apparente puisqu'on identifie toujours les arbitrages à partir des bornes des fenêtres des opérations et qu'il est de même possible d'obtenir des ajustements à partir des arbitrages détectés.

Dans notre contexte, l'appel *Appel* $(F_i^{P_k}, F_j)$ entre les deux formateurs est formalisé, dans la communauté d'ordonnancement, par une *arête disjonctive notée* $[i^{P_k}; j]$. On appelle *clique de disjonctions* un ensemble d'opérations L tel que toute paire d'opérations extraite de L correspond à une arête disjonctive.

Soit θ l'ensemble d'appels effectuées par les deux formateurs et $T^t_{i \rightarrow j}$ (il est possible que $i = j$) la durée de traitement sous-jacent à l'appel *Appel* $(F_i^{P_k}, F_j)$

[2]Voir par exemple [Riv99].

effectué à l'instant t. On associe de même aux opérations une fenêtre d'exécution $[r^t_{i\rightarrow j}; d^t_{i\rightarrow j}]$ correspondant à une contrainte implicite ou explicite. Si l'on s'en tient à ces seuls éléments, une solution au problème sera constitué par la donnée d'un ensemble $t = (t_z)_{z\in\theta}$ vérifiant :

- $\forall z \in \theta, r^t_z \leq t_z \leq d^t_z - T^t_z$,
- $\forall s \in [1, L], \forall [z, w] \in K_s, t_z + T^t_z \leq t_w \vee t_w + T^t_w \leq t_z$.

Comme nous le mentionnons ci-dessus, pour mieux délimiter l'espace des solutions, les opérations locales opèrent au niveau des cliques de disjonctions par : arbitrage et/ou réduction de domaine. Par la suite nous présentons comment nous appliquons la technique des arbitrages dans notre contexte.

Arbitrage

Cette technique consiste à identifier les relations de précédence entre tous les appels. Plusieurs variantes ont été proposées, par exemple dans les arbitrages sur des ensembles ascendants/descendants [CP90], nous distinguons trois cas ($J \subseteq K, op \notin J$) :

- L'opération *op* (par exemple un *appel* $({F_i}^{P_K}, F_j)$) ne peut être ordonnancée *avant* l'ensemble J si :

$$r_{op} + T_{op} + T(J) > d(J) \tag{6.1}$$

- L'opération *op* ne peut être ordonnancée *au milieu* de l'ensemble J si :

$$r(J) + T_{op} + T(J) > d(J) \tag{6.2}$$

- L'opération *op* ne peut être ordonnancée *après* l'ensemble J si :

$$r(J) + T(J) + T_{op} > d_{op} \tag{6.3}$$

Lorsque les conditions (6.1) et (6.2) sont vérifiées, l'opération i doit nécessairement s'exécuter après J : on dit qu'il y a arbitrage sur ensemble ascendant entre i et J. De façon symétrique, si les conditions (6.2) et (6.3) sont satisfaites, l'opération i doit alors s'exécuter avant J. On parle dans ce cas d'arbitrage sur ensemble descendant.

6.7.2.5 Synthèse sur la composition en parallèle

Dans cette section, nous avons proposé de composer les formateurs en parallèle. Ceci revient à résoudre un problème d'ordonnancement des appels entre les deux formateurs.

L'application des techniques d'ordonnancement dans notre contexte est limitée par trois facteurs :

- Nous ne pouvons faire aucune hypothèse sur les langages de présentation. En effet, la majorité de ces langages sont des standards ouverts et le vocabulaire peut évoluer constamment alors que les techniques des arbitrages ont pour principe de base l'étude des liens entre toutes les opérations à ordonnancer (par exemple, les liens entre tous les appels *appel* $({F_i}^{P_K}, F_j)$),
- Le fait d'étudier les liens entre tous les éléments du langage source et le langage XEF va à l'encontre de notre ambition initiale qui consiste à proposer une architecture de couplage indépendante du langage source,

– Résoudre un problème d'ordonnancement est souvent une tâche complexe et fastideuse, et d'autant plus difficile à traiter dans notre contexte. En effet, dans les problèmes d'ordonnancement « traditionnels », on considère que le temps d'exécution de chaque tâche est connu alors que dans notre contexte, le temps sous-jacent à chaque appel est inconnu.

Dans l'état actuel de la recherche, nous avons donc montré l'insuffisance des techniques d'ordonnacement pour répondre aux besoins de couplage des formateurs.

Nous ne sommes bien sûr pas entrés dans les détails de chaque étape d'ordonnancement. Nous voulions insister sur le fait que le problème d'ordonnancement est étudié depuis longtemps dans des domaines divers, notamment dans les systèmes informatiques et dans les systèmes de production. Seule l'importance relative de telle ou telle contrainte change alors (les caractéristiques des ressources, les contraintes portant sur les opérations,...) mais pas la façon de les aborder ni l'ensemble des outils nécessaires qui relèvent de toute évidence de la panoplie des méthodes de résolution des problèmes combinatoires. Cependant, il n'existe pas, à ce jour, de méthodologie claire pour répondre à ce problème.

6.7.3 Composition en série

Nous avons conclu dans la section précédente qu'il n'est pas envisageable d'opter pour une composition en parallèle. Pour faire face à la complexité inhérente à la gestion de communication entre les formateurs, nous proposons une architecture de couplage en série.

L'idée clé de cette approche est de minimiser les interactions entre les deux formateurs en séparant les deux processus de formatage. Chaque formateur une fois lancé doit formater tout le document avant de passer la main à l'autre formateur. Le processus de formatage peut éventuellement comprendre plusieurs appels aux deux formateurs (cf.FIG.6.9). Avant de détailler le mécanisme et l'architecture complète du système que nous proposons pour la mise en œuvre de cette approche, nous donnons dans le paragraphe qui suit son principe général.

6.7.3.1 Principe

Cette approche consiste à lancer l'un des deux formateurs après l'exécution du second formateur, avec éventuellement une répétition de ces deux opérations. Le processus de formatage peut être donné par la formule :

$$(F = F_{source} \oplus F_{XEF}) \Leftrightarrow (Formatage_{source} \mid Formatage_{XEF})^{+} \qquad (6.4)$$

– Le symbole « | »indique que l'ordre n'est pas important,
– Le symbole « + »spécifie que l'ensemble $\{Formatage_{source} \mid Formatage_{XEF}\}$ peut contenir un ou plusieurs appels,
– $Formatage_{source}$ désigne le processus de formatage du document source entier.

Contrairement à la composition en parallèle, la granularité n'est plus la propriété (P_k) mais l'ensemble de toutes les propriétés (tout le document).

La formule (6.4) ne spécifie pas lequel des deux formateurs sera lancé le premier. Le formatage peut commencer par un formatage du document source et ensuite un formatge XEF (cf.FIG.6.9.A) tout comme il peut commencer par un formatage XEF (cf.FIG.6.9.B).

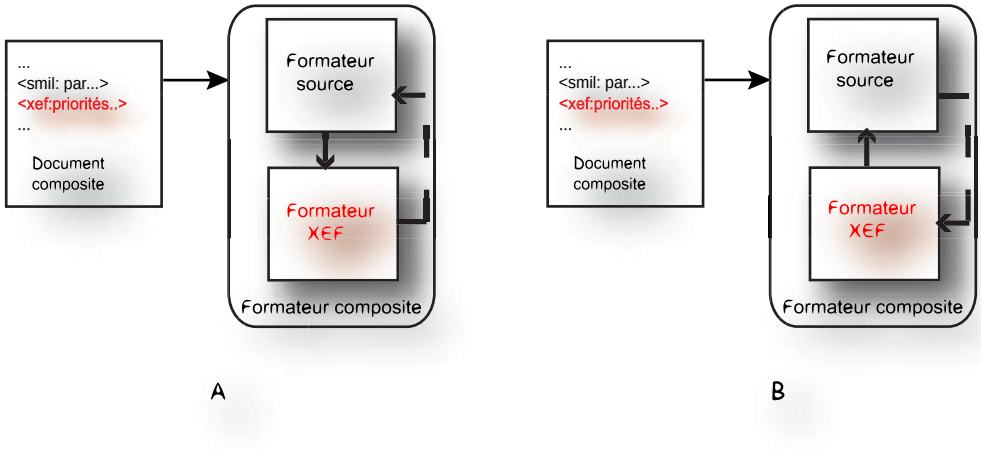

FIG. 6.9 – Composition en série des formateurs

6.7.3.2 Architecture générale du système de composition en série

Conformémment à la démarche de conception suivie dans la programmation orientée composant et pour répondre aux besoins identifiés ci-dessus, nous avons donc définit l'architecture illustrée dans la figure FIG.6.10.

Le mode de couplage au niveau des langages est celui du couplage sur l'instance formatée du document source présenté dans le paragraphe (§.5.6). Ce choix nous a conduit à spécifier les quatre modules suivants : le formateur source, le formateur XEF, un connecteur entre ces deux formateurs et le formateur composite qui supervise tout le système.

6.7.3.2.1 Formateur source

Le formateur source prend en charge le formatage du document source qui contient le scénario spatio-temporel.

Comme nous le mentionnons dans le paragraphe (§.6.6.5), le formateur comprend deux modules :

- *Frontend*[3] : ce module charge et analyse le document source en créant un arbre DOM.
- *Moteur de formatage* : ce module calcule une représentation de bas niveau des attributs de l'arbre DOM.

[3]La traductions françaises *frontal* ou *surcouche* [Lex] n'etant pas reconnues, nous continuons d'utiliser le terme anglais dans la suite de ce rapport

6.7.3.2.2 Formateur XEF

Nous avons introduit dans le paragraphe (§.5.6) la structure du document de contrôle. Celui-ci contient en plus des contrôles XEF, des expressions de chemin (exprimées dans un langage de sélection). L'arbre DOM généré à la fin de l'analyse (la sortie du module *frontend*) contient donc les éléments XEF et les sélecteurs.

Le moteur de formatage XEF doit pouvoir récupérer les éléments du document source désignés par les expressions de chemins pour y appliquer les contrôles associés. Ce rôle est assumé par le *connecteur source/XEF*.

6.7.3.2.3 Connecteur source/XEF

En particulier, le module d'interfacage est composé de deux parties :

- *Extraction* : ce module permet d'extraire les nœuds de l'arbre source sur lesquels l'auteur veut rajouter des contrôles. Ces derniers sont identifiés grâce aux expressions de chemin du document de contrôle (la flèche 1 de la figure FIG.6.10). Suivant le protocole de coopération entre les formateurs (cf.§.6.7.3.3.2), l'extraction peut se faire sur les nœuds non formatés (la flèche 2 de la figure FIG.6.10) ou sur des nœuds formatés (la flèche 3 de la figure FIG.6.10),
- *Traduction* : il assure la traduction des nœuds sélectionnés dans l'étape précédente en des nœuds XEF (la flèche 4 de la figure FIG.6.10). Après le formatage XEF, ce module assure la traduction de ces nœuds en des nœuds source (la flèche 5 de la figure FIG.6.10).

Suivant la technique mise en œuvre pour assurer la cohérence du scénario (cf.§.6.7.3.3), l'extraction peut être partielle ou transitive.

6.7.3.2.4 Formateur composite

Le rôle du formateur composite est, d'une part, la gestion de la communication entre les deux formateurs. Ce rôle est assuré par le module *gestionnaire de la communication* de la figure FIG.6.10 (cf.§.6.7.3.3) et d'autre part, le maintien de la cohérence du scénario (cf.§.6.7.3.4) .

6.7.3.3 Gestion de la communication

La gestion de la communication entre les deux formateurs comprend en plus de la définition du protocole de coopération entre les formateurs, la spécification des interfaces de communication. Comme nous avons vu dans le chapitre 3, les interfaces doivent être définies de manière indépendante à toute implémentation pour que l'architecture de couplage soit réutilisable pour différents formateurs.

6.7.3.3.1 Définition des interfaces

Nous avons montré dans le paragraphe (§.6.5) que les besoins de couplage du formateur XEF avec d'autres formateurs nous obligent à travailler sur des formateurs qui génèrent un arbre DOM.

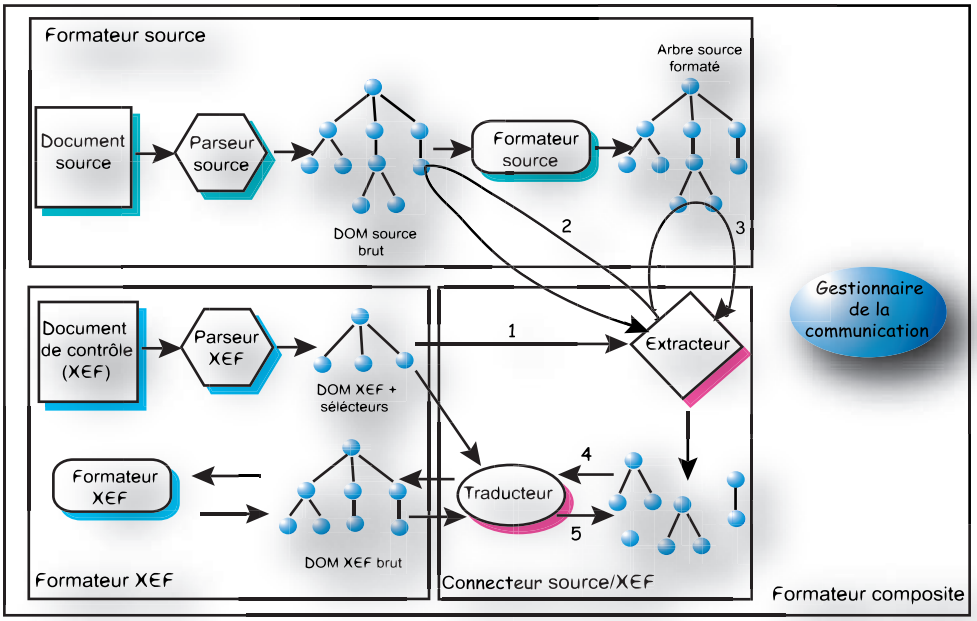

FIG. 6.10 – Architecture du système de composition en série des formateurs

Nous avons profité de cette contrainte pour définir des interfaces génériques. Les deux formateurs communiquent donc via leurs arbres DOM. Ce type de couplage nous offre les avantages suivants :

- Un couplage faible entre les formateurs (couplage de données),
- Le **standard** DOM permet l'indépendance de l'architecture de couplage par rapport à l'architecture des formateurs,

Cependant, comme évoqué précédemment le formateur génère deux types d'arbre DOM. Un arbre DOM brut généré après la phase d'analyse (parsing) et un arbre DOM formaté. La question est de choisir entre l'un des ces deux arbres.

Il est clair que la réponse à cette question dépend du mode de coopération entre les formateurs.

6.7.3.3.2 Protocole de coopération

Le protocole de coopération consiste à définir :

- l'ordre d'exécution des formateurs,

– le type de l'arbre DOM sur lequel est réalisé le couplage.

Ces deux choix ne sont pas indépendants et doivent être traités conjointement. Nous distinguons deux cas de figure :

– *Couplage sur l'arbre DOM brut* : dans cette approche, le couplage s'effectue sur l'arbre DOM généré après le parsing de document source par le formateur source. Le formateur XEF récupère et formate les nœuds sur lesquelles on a spécifié des contrôles.
  Ce type de couplage est utile dans le cas des formateurs qui ne formate pas ou formate partiellement l'arbre DOM. L'intérêt de cette architecture est que la cohérence du scénario est maintenu par le formateur source. En effet, le fait que les valeurs des propriétés ne sont calculées qu'au moment de l'exécution permet de garder les liens existants entre les éléments et donc après le formatage XEF le formateur source peut gérer la cohérence du document.
  Cependant ce type d'architecture ne permet pas de profiter du formateur source et le formateur XEF doit lui même calculer toutes les propriétés dont il a besoin.
– *Couplage sur l'arbre DOM formaté* : le formateur XEF est lancé après le formateur source. Dans ce cas le couplage s'effecctue sur l'arbre DOM formaté par le formateur source.
  Le fait de coupler les deux formateurs via l'arbre DOM formaté pose deux problèmes :
  • la validité de document par rapport à la spécification,
  • le problème des incohérences que peut entraîner la modification de l'arbre DOM par le formateur XEF.

6.7.3.4 Gestion de la cohérence

Le principe de cette approche est de ne manipuler que les nœuds de l'arbre source sur lesquels on veut rajouter des contrôles :

$$Formatage_{XEF} \Leftrightarrow \begin{cases} S_{contrôle} : \textit{les noeuds contrôlés du document source} \\ \forall e \in S_{contrôle} \begin{cases} Extraction(e), \\ Traduction(e), \\ Formatage_{XEF}(e) \end{cases} \end{cases}$$

Dans cette approche, le formateur XEF a un contrôle local sur les éléments. Les éléments (re)formatés par XEF et ayant des dépendances avec d'autres éléments risquent de poser un problème d'incohérence. Notons S_d l'ensemble des éléments du document source ayant des dépendances avec d'autres éléments, alors :

$$\begin{cases} S_d = \{e_1, ...e_n\} \mid (n \in \mathrm{N} \wedge \forall i \in \{1...n\}, e_i \rightsquigarrow S_d^{e_i}) \\ S_d^{e_i} \textit{ est le sous ensemble de } S_d \textit{ contenant des éléments ayant des relations avec } e_i \\ \textit{Si } (S_{contrôle} \cap S_d) \neq \emptyset \Rightarrow \textit{risque d'incohérence} \end{cases}$$

Dans le cas où le formateur source calcule les dépendances, les propriétés devraient être recalculées. cependant s'il ne calcule pas les dépendances le système n'as pas besoin de recalculer les valeurs des propriétés. Par la suite, nous ne considérons que le premier cas.

Le formateur F doit donc être doté d'un mécanisme permettant de détecter et de traiter ces incohérences. Pour ce faire, nous distinguons deux variantes :

6.7.3.4.1 Extraction partielle

Une première idée intuitive pour assurer la cohérence du scénario est de relancer le formateur F_{source} pour reformater l'arbre du document. Dans ce cas, le formateur F_{source} recalcule les valeurs de toutes les propriétés. En particulier, il peut exister un ensemble de propriétés $p^k_{e_i}$ tel que :

$$\begin{cases} \forall e_i \in E_{source}, \exists p^k_{e_i} \in P_{e_i} \mid formatage_{XEF}(p^k_{e_i}) \wedge formatage_{source}(p^k_{e_i}) \\ E_{source} \text{ est l'ensemble des éléments du langage source} \\ P_{e_i} \text{ est l'ensemble de propriétés de l'élément } e_i \end{cases}$$

Le fait que les valeurs de ces propriétés ont été recalculées par le formateur F_{source} peut lui même entraîner des incohérences dans le scénario. Le formateur F relance donc le formateur F_{XEF} une autre fois. De façon générale, le processus de formatage se décline en une boucle de $formatage_{source}$ et de $formatage_{XEF}$.

Détection de la terminaison du formatage

Pour répondre au problème de bouclage, il faut doter le système d'un mécanisme permettant de détecter la fin du formatage.

Nous définissons deux variables $EtatPrécédentFormateur$ et $EtatActuelFormateur$ qui gardent une image des différentes propriétés à la fin de chaque étape de formatage (c'est-à-dire après chaque appel d'un formateur). Le processus de formatage s'achève quand :

$$EtatPrécédentFormateur = EtatActuelFormateur$$

Nous présentons dans la figure FIG.6.11 l'algorithme qui permet de réaliser cette fonction. L'architecture présentée dans la figure FIG.6.10 est enrichie par un nouveau module (gestionnaire de la cohérence) qui implémente cet algorithme. Ce module est implémenté au niveau du formateur composite F (cf.FIG.6.12).

Le processus de détection de la terminaison du formatage peut être amélioré comme suit : au lieu de garder l'état de toutes les propriétés après chaque formatage, nous proposons de raffiner l'architecture précédente en ajoutant *un gestionnaire des mises à jour*. Le rôle de ce module est de propager les mises à jour après chaque formatage de l'arbre formaté vers les nœuds sélectionnés pour que l'arbre DOM manipulé par l'autre soit à jour. S'il constate qu'il n'y a plus de mise à jour, le gestionnaire des mises à jour termine le processus de formatage.

Cette algorithme ne garantit pas l'existance d'une solution et le processus de formatage peut en particulier entrer dans une boucle infinie. En effet, les propriétés calculées par un formateur F_i peuvent être remises en cause par l'autre formateur F_j. Ce problème est dû au fait qu'on a pas un contrôle sur les formateurs F_i et F_j (on ne peut pas forcer un formateur à prendre les valeurs des propriétés déjà calculées par l'autre formateur).

$(i, j) \in \{source, XEF\}; i \neq j;$

$EtatPrécédentFormateur = EtatActuelFormateur = \emptyset;$

$formatage_i;$

$EtatActuelFormateur \leftarrow Sauvegarder\_Etat\_Propriété;$

$formatage_j;$

$EtatPrécédentFormateur \leftarrow EtatActuelFormateur;$

$EtatActuelFormateur \leftarrow Sauvegarder\_Etat\_Propriété;$

Tant que($EtatPrécédentFormateur \neq EtatActuelFormateur$)

{

$formatage_i;$

$EtatPrécédentFormateur \leftarrow EtatActuelFormateur;$

$EtatActuelFormateur \leftarrow Sauvegarder\_Etat\_Propriété;$

Si ($EtatPrécédentFormateur \neq EtatActuelFormateur$)

$formatage_j;$

$EtatPrécédentFormateur \leftarrow EtatActuelFormateur;$

$EtatActuelFormateur \leftarrow Sauvegarder\_Etat\_Propriété;$

}

$Retourne$(formatage terminé);

FIG. 6.11 – Algorithme de détection de la fin du formatage

6.7.3.4.2 Extraction transitive

Nous venons de montrer que s'il est facile à mettre en oeuvre, le formatage en appel successif des formateurs : $formateur_{source}$ et $formateur_{XEF}$ soulève un sérieux problèmes de bouclage infini. Nous proposons dans ce paragraphe une variante permettant d'éviter le problème de cycle.

A la différence de la technique précédente, nous proposons de traduire non seulement les éléments e_i sur lesquels on a appliqué un contrôle XEF mais en plus de ces éléments, on traduit tous les éléments qui en dépendent $S_d^{e_i}$ (cf.FIG.13).

$$Formatage_{XEF} \Leftrightarrow \begin{cases} S_{contrôle} : \textit{les noeuds contrôlés du document source} \\ \forall e_i \in S_{contrôle} \begin{cases} Extraction\{e_i, S_d^{e_i}\}, \\ Traduction\{e_i, S_d^{e_i}\}, \\ Formatage_{XEF}\{e_i, S_d^{e_i}\} \end{cases} \end{cases}$$

Cette variante évite les appels successifs aux deux formateurs et le problème de bouclage ne se pose donc plus. Cependant, la traduction transitive peut

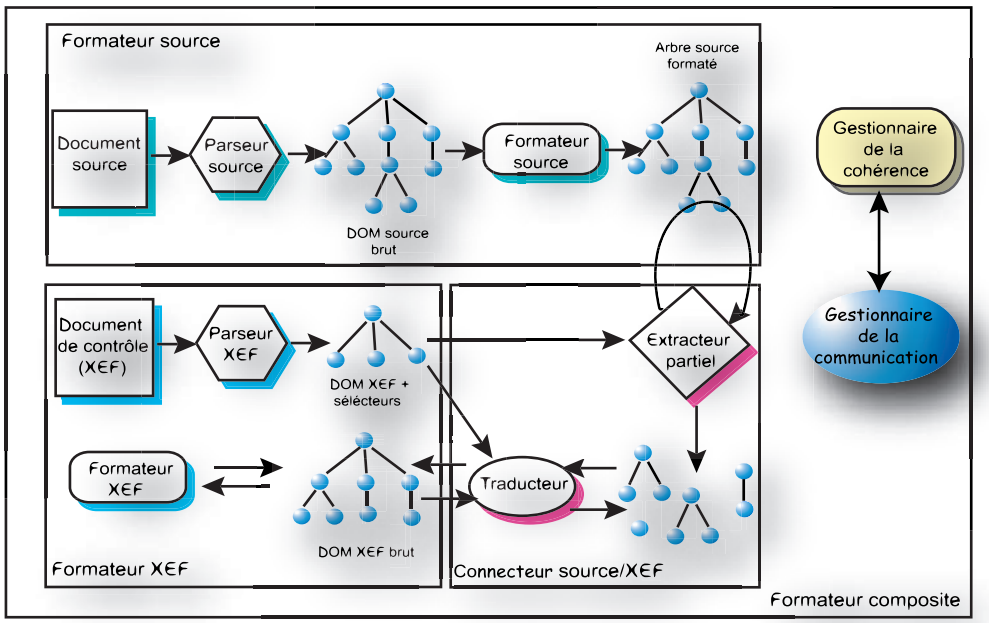

FIG. 6.12 – Architecture du système de composition en série des formateurs - extraction partielle

conduire à une traduction totale (tout l'arbre soure est traduit en un arbre XEF) et donc on profite pas du formateur source.

6.8 Bilan

Dans ce chapitre nous avons présenté deux approches pour le couplage des formateurs.

Nous avons montré que dans la composition en parallèle la problèmatique est d'arriver à ordonnancer les appels entre les deux formateurs.

L'ordonnancement des appels est un processus NP-difficile, nous avons donc proposé la composition en série comme une alternative. Dans cette approche, la granularité n'est plus la propriétés mais tout le document. le problème auquel nous étions confrontés dans cette approche est la gestion de la cohérence.

Dans une première variante nous avons proposé une politique d'extraction partielle des nœuds avec des appels successifs entre les deux formateurs. Nous avons proposé une deuxième variante qui se base sur la traduction transitive des nœuds. Dans cette approche, il y'a risque de traduction totale et donc on ne peut pas profiter de l'autre formateur.

Concernant les formateurs qui ne calculent pas les valeurs des propriétés,

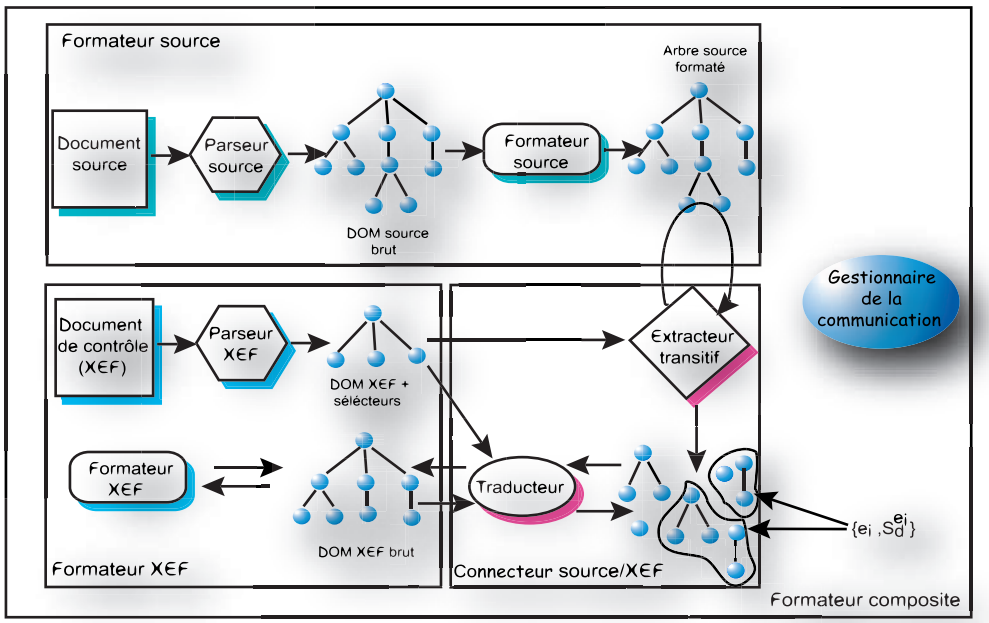

FIG. 6.13 – Architecture du système de composition en série des formateurs - extraction transitive

nous avons donc proposé de ne traduire que les nœuds contrôlés du document source. la cohérence du scénario est confiée au formateur source. En effet, les dépendances entre les éléments ne sont pas calculées et le formateur source garde la sémantique de ces dépendances, c'est au moment de l'exécution que ces dépendances sont résolues. Par exemple, après un formatage XEF, le formateur SMIL de l'environnement XSmiles (qui ne résoud pas les dépendances entre les éléments du document) va exploiter les valeurs des propriétés calculées par le formateur XEF pour exécuter le scénario. Le traitement des incohérences dépend de la spécification du langage source. Dans le cas de SMIL par exemple, si un traitement XEF supprime un élément référencé par d'autres éléments, alors la recommandation SMIL propose d'ignorer tous les attributs référençant cet élément.

Remarquons que dans cette solution nous n'avons pas un contrôle total sur le formatage du document source. En particulier le formateur peut entrer dans une situation d'échec. Ce type d'architecture ne répond donc pas à nos besoins de couplage du formateur XEF. On ne peut profiter du formateur XEF que dans le cas où le formateur source formate effectivement le document source.

Concernant l'ordre d'exécution des formateurs, il est serait intéressant de lancer le formateur XEF en premier lieu pour formater les propriétés globales. En effet, les propriétés globales permettent de spécifier des critères dont les

autres éléments du document dépendent. Par exemple, le fait de spécifier la durée totale de la présentation peut remettre en cause les durée de tous les éléments du document. Par contre, dans le cas des éléments XEF ayants un contrôle local sur les éléments du document source (par exemple : les alternatives et les replis) nous avons proposé de lancer le formateur XEF après le formateur source. Cependant, nous pensons qu'une étude plus détaillée est nécessaire pour définir un ordre d'exécution des éléments XEF en tenant compte des spécifications de chaque langage source.

Chapitre 7

Mise en œuvre

7.1 Introduction

Dans ce chapitre nous présentons la mise en œuvre que nous sommes entrain de faire de notre proposition. Celle-ci est en cours d'implémentation dans l'environnement Limsee 1.0 [LimSee1], un système d'édition et de présentation de documents multimédia.

Rappelons que notre objectif est d'une part tester l'architecture que nous proposons pour le couplage et d'autres part, valider le langage XEF.

L'architecture est celle présentée dans la figure FIG.6.12. Celle-ci comprend deux parties :
- implémentation d'un connecteur XEF/source,
- implémentation du formateur composite.

7.2 Implémentation du connecteur XEF/source

L'implémentation de ce connecteur consiste à implémenter :
- un module de « extraction »,
- un module de « traduction ».

7.2.1 Module « extraction »

Nous avons implémenté le module « extraction »responsable de l'extraction des nœuds du document source sur lesquelles des contrôles XEF ont été spécifiés dans le document de contrôle. Pour ce faire, ce module se sert des expressions de chemin du document de contrôle pour sélectionner les nœuds en question.

Le module d'extraction s'occupe aussi de l'opération inverse. C'est-à-dire après que les nœuds extraits sont traités par le formateur XEF, il seront réinjectés dans l'arbre DOM source.

Remarquant que dans cette approche les éléments ayants des relations avec les nœuds déjà séléctionnés ne seront pas traités.

1. <seq>
2. <par id="par1">
3. ...
4. </par>
5. ...
6. <video id="A" src="toto.au" begin="par1.end"/>

A-Document source

1. <//seq/par :{repli="réduction (p1)"}>
2. ...

B-Document de contrôle

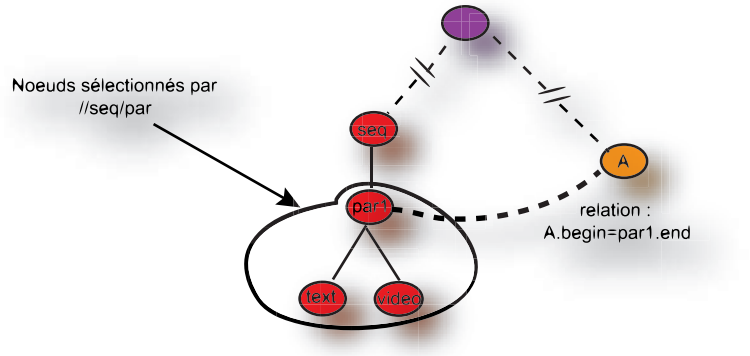

FIG. 7.1 – Exécution du module « extraction »

Exemple

Nous donnons dans la figure FIG.7.1 un exemple d'un contrôle XEF placé sur un ensemble d'éléments d'un document SMIL (cf.FIG.7.1.A). Ces éléments sont identifiés par une expression de chemin du langage XPath (cf.FIG.7.1.B).

L'expression XPath *//seq/par* permet de sélectionner l'élément *par*1. Malgré que ce dernier est lié à lélément *A* par la relation *begin* = "*par*1.*end*", l'élément *A* ne sera pas sélectionné.

7.2.2 Module « traduction »

Les nœuds extraits dans l'étape précédente doivent être transformés en des nœuds XEF pour que le formateur XEF puisse les traiter. Inversement, les nœuds traités par le formateur XEF doivent être traduits en des nœuds du langage source.

7.3 Implémentation du formateur composite

L'implémentation de ce module comprend la réalisation de deux fonctions :
- la gestion de la communication entre les deux formateurs,
- la gestion de la cohérence.

7.3.1 Module « gestionnaire de la cohérence »

Nous avons implémenté l'algorithme présenté dans la figure FIG.7.11 responsable de la gestion de la cohérence du scénario. Lorqu'il reçoit un message en provenance du formateur actif indiquant la fin de sa tâche, il compare l'état de toutes les propriétés avant et après le formatage. S'il constate un changement dans les valeurs des propriétés alors le processus de formatage n'est pas terminé et il passe la main au module de synchronisation.

7.3.2 Module « gestionnaire de la communication »

Ce module est chargé de la gestion des appels entre les deux formateurs. Lorsqu'il reçoit un message du gestionnaire de la cohérence indiquant que le processus n'est pas achevé, il relance l'un des deux formateurs.

Dans cette implémentation le processus de formatage commence par le formateur source et ensuite le formateur XEF.

7.4 Présentation de l'environnement Limsee

LimSee est développé par l'equipe OPERA. C'est un éditeur de structures temporelles et spatiales pour les documents multimédia spécifiés dans le langage SMIL. Il propose à l'auteur une visualisation de son scénario temporel ainsi que son édition par manipulation directe.

Le principe de LimSee repose sur l'utilisation de plusieurs vues synchronisées sur une structure de données interne commune. Les principales vues sont :
- *une vue hiérarchique* : cette vue offre une visualisation classique sous la forme d'un arbre du document (cf.FIG.7.1.A),
- *une vue temporelle* : cette vue est une représentation fidèle du comportement temporel du document (cf.FIG.7.1.B)
- *une vue attribut* : cette vue est un peu particulière, ce n'est pas une vue au même sens que les autres. Elle permet d'afficher des formulaires textuels contenant les valeurs des attributs (cf.FIG.7.1.C)

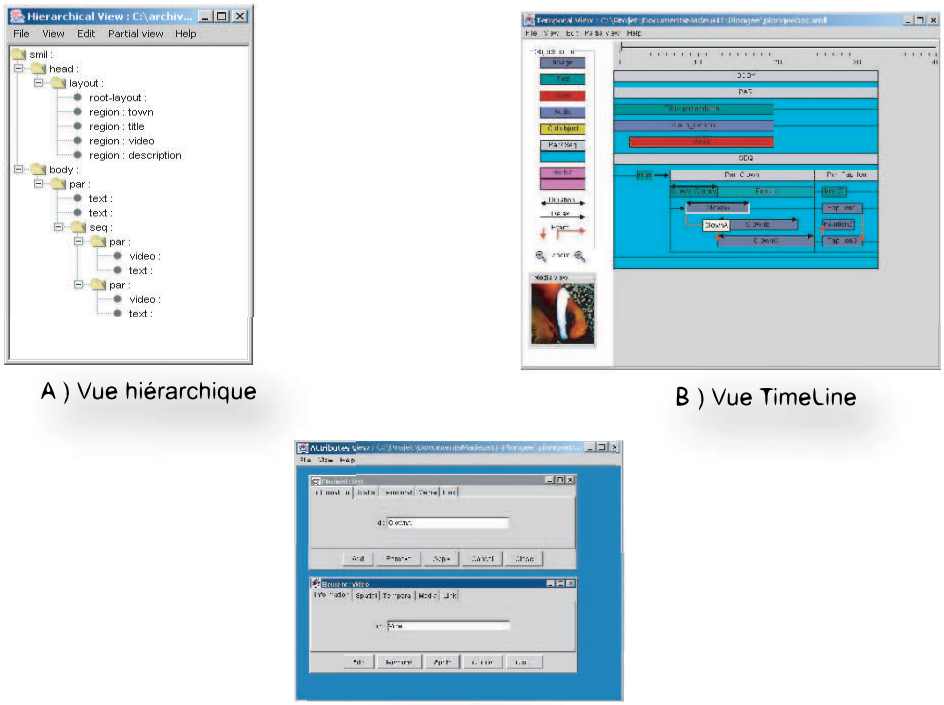

FIG. 7.2 – Les différentes vues dans LimSee

A travers chaque vue l'auteur dispose des opérations d'édition de base Copier/Coller Ajout/Suppression d'objets. En plus de ces opérations d'édition, la vue Temporelle offre des techniques de manipulation directe des objets graphiques représentant les différents médias du document ainsi que leurs attributs temporels.

LimSee tente de se rapprocher des techniques WYSIWYG non seulement par un placement précis des médias dans la vue temporelle mais aussi et surtout en propageant immédiatement á l'ensemble du document toute action de l'auteur visant à modifier le scénario temporel. Pour ce faire, LimSee utilise les techniques de propagation de contraintes. La tâche de l'auteur se trouve donc simplifiée lors de l'édition d'un document puisqu'il ne doit plus se préoccuper de recalculer ou de repositionner ses média lors de l'ajout ou du retrait d'un élément ou plus simplement lors de la modification d'un instant de début, de fin ou de durée, le système effectue ce travail à sa place.

LimSee génère et manipule une structure interne sous forme d'un arbre DOM.

7.5 Protocole de test

Le document utilisé pour le test est le document SMIL présenté dans la figure FIG.5.1. Les contrôles XEF visent à répondre aux besoins de présentation donnés dans le paragraphe §.5.2.2.3.

Conclusion

Rappel des objectifs

Le travail mené durant ce projet de DEA se situe à l'intersection de deux domaines : le formatage des langages de présentation et le génie logiciel, en particulier la composition des architectures logicielles.

Ces dernières années, plusieurs langages de présentation ont émergé pour faire face aux défis que constitue l'explosion fulgurante du Web et le déploiement de grandes variétés de terminaux de restitution. Cependant, ces langages comme leurs formateurs n'offrent pas de solutions complètement satisfaisantes et peuvent amener à des situations d'échec.

La démarche suivie jusqu'à présent pour combler les lacunes des langages de présentation est une démarche radicale basée sur la spécification d'un nouveau langage. Le travail mené dans [BR02,BR02+] propose une solution innovante pour éviter les cas d'échec de formatage. L'idée directrice est d'étendre l'expressivité des langages actuels en ajoutant des capacités de contrôle qui puissent être pris en compte par le processus de formatage plutôt que de spécifier un nouveau langage à partir de zéro. Le but est non seulement d'intégrer ces contrôles dans les langages existants mais également de proposer des services de formatage visant à intégrer leur traitement dans les formateurs existants. C'est à cet aspect que nous nous sommes intéressés.

Synthèse

Dans ce rapport, Nous avons étudié le processus de couplage de deux langages de présentation en vue de réaliser un système de formatage plus performant. Ce couplage s'applique à deux niveaux, le couplage au niveau des langages et le couplage au niveau des formateurs.

L'état de l'art dressé dans la première partie nous a amené à la constatation d'une part qu'il existe un certain nombre de techniques pour le couplage des langages basés sur XML et que d'autre part, il n'existe pas - si on fait abstraction de quelques travaux sur la composition des applications centrées documents - de travaux portant sur la composition des formateurs XML.

Nous avons alors proposé une architecture logicielle permettant le couplage d'un langage de contrôle de formatage et des services de formatage associés avec des systèmes de formatage existants. Notre contribution concerne à la fois

le couplage au niveau des langages, et celui des traitements (formateurs).

Nous avons étudié dans le chapitre 5 plusieurs propositions pour le couplage de deux langages de présentation. Une contrainte importante pour nos besoins était que le couplage devrait permettre la modification de l'arbre du document. La conclusion de cette étude est que seulement les techniques de couplage par transformation vers un document composite et le couplage sur l'instance formatée du document source satisfont ce besoin.

Concernant la composition d'architectures logicielles, divers travaux sont en cours pour permettre la composition à travers les différents paradigmes de la programmation. Néanmoins, c'est le paradigme de la programmation par composant - et plus récemment la programmtion par aspect - qui propose des techniques intéressantes permettant la composition des logiciels. Ainsi, plusieurs techniques de composition ont émergé. Citons par exemple la composition structurelle, la composition fonctionnelle et la composition comportementale. Cependant, aucune solution définitive ne se dégage actuellement. En effet, la mise en oeuvre de la composition se trouve confrontée à différents problèmes et notamment la définition des interfaces génériques et la gestion de la cohérence des données échangées entre les composants.

En s'inspirant des techniques de composition mises en œuvre dans le paradigme de la programmtion par composant, nous avons donc proposé dans le chapitre 6 une architecture logicielle permettant le couplage des formateurs en prenant en compte les deux techniques de couplage citées ci-dessus. La seule hypothèse faite sur les formateurs est qu'ils génèrent un arbre DOM.

Évaluation

Vu l'avantage que présente la composition d'architectures logicielles sur le plan du coût, il serait intéressant d'appliquer cette technique dans le domaine du génie documentaire pour la création de formateurs plus performants en profitant des formateurs existants.

Bien que les formateurs actuels ne sont pas développés dans l'esprit d'être coupler les uns aux autres, l'aspect modulaire et déclaratif des langages associés le permet.

L'architecture que nous avons spécifié dans le chapitre 6 propose un couplage faible entre les deux formateurs, ce qui mène à une architecture générique pour le couplage des formateurs. Le couplage s'effectue par l'intermédiaire des arbres DOM (couplage de données) manipulés par les formateurs.

L'avantage de cette technique de couplage tient essentiellement au fait que DOM est un standard. Le couplage via des données dont la structure est standardisée assure que le système de couplage est réutilisable pour différents formateurs. Le système de couplage est générique en ce sens qu'il permet le couplage des formateurs indépendamment des données qu'ils manipulent. C'est ainsi, que même si nous nous sommes basés dans notre étude sur le langage SMIL, le système de couplage est indépendant du langage source.

Cependant, comme nous avons vu dans le chapitre 6, il existe des formateurs qui ne génèrent pas des structures DOM mais il manipulent des structures de

données propres, ce qui est le cas des formateurs utilisant SAX et des techniques ad hoc. Le système que nous proposons se trouve donc incapable de coupler des formateurs qui ne manipulent pas une structure DOM.

Concernant le langage XEF, malgré le fait que la mise en œuvre du système de couplage ne soit pas encore terminée, nous avons eu l'occasion à plusieurs reprises dans ce rapport de mettre en évidence l'intérêt de l'approche adoptée dans le langage XEF. En particulier, d'une part, le langage XEF étend le pouvoir d'expression des langages de présentation et d'autre part, le formateur XEF permet un contrôle plus subtil du comportement des autres formateurs.

Perspectives

Il est clair que ce travail ne constitue qu'un point de départ pour des recherches futures. Nous donnons dans ce paragraphe quelques voix d'investigation qui nous paraissent intéressantes et qui pourraient guider des travaux futurs.

Le couplage par transformation vers un document composite

La technique du couplage au niveau des langages utilisée dans l'architecture présentée dans le paragraphe §.6.7.3.2 est celle du couplage sur l'instance formatée du document source, nous souhaiterions étudier les modifications à apporter à cette architecture pour pouvoir supporter la technique du couplage par transformation vers un document composite.

Composition en parallèle

Nous avons montré dans le chapitre 6 que la composition en parallèle pose un problème d'ordonnancement des appels NP-difficile. Il nous semble intéressant d'essayer d'appliquer ce type de composition en tenant compte des particularités des langages sources manipulés. En particulier, nous envisageions d'étudier la possibilité de trouver un ordre d'exécution entre les éléments XEF et un sous-ensemble du langage source. Par exemple, est-il possible de trouver un ordre d'exécution entre les éléments XEF et les éléments du langage SMIL 1.0 qui n'autorise que des liens entre des cousins ?

Composition des formateurs basés sur SAX

Le système que nous avons proposé ne supporte que les formateurs basés sur DOM, nous envisagerions d'étudier la composition des formateurs basés sur SAX. En particulier, nous pensions que le couplage par transformation vers un document composite au niveau des langages peut, conjointement avec un couplage des formateurs SAX, constituer un système de couplage complet.

Langage de composition de formateurs

Nous nous sommes fixés comme objectifs non seulement de faciliter la tâche de l'auteur en proposant un mécanisme simple pour le couplage au niveau des langages mais aussi de rendre le processus de couplage des formateurs le plus simple possible.

Il peut être intéressant de spécifier un langage de composition de formateurs. Nous proposons d'étudier la possibilité de spécifier la composition des formateurs dans une application XML. Quelques travaux ont tenté d'appliquer cette approche dans la composition des composants [Bir01].

Notre idée est de définir une application XML *XFoCL* (XML Formaters Composition Language) permettant aux développeurs de décrire le schéma de couplage des formateurs. Nous donnons dans la figure ci-dessus la structure générale de ce que peut être un document XFoCL. Nous souhaiterions étudier les propriétés de chaque élément (formaters, formater, connector). Par exemple, dans la figure nous avons définit deux propriétés *id* et *src* pour l'élément *formater* mais une étude plus profonde est nécessaire pour définir le vocabulaire du XFocl.

```
1. <xfocl>
2.  <formaters>
3.   <formater id="identificateur" src="...">
4.   ...
5.  </formaters>
6.  <connector source="..." target="..." interface="..."/>
7. </xfocl>
```

Fig. 7.3 – Structure d'un document XFoCL

Annexes

Annexe 1

La spécification du document SMIL présenté dans la figure FIG.5.1 en Madeus

```
1. <madeus>
2.  <media>
3.   <text id="DescriptionFilm" >Titre : TITANIC ; Pays : Etats unis
4.   </text>
5.   <text id="DescriptionActeur" > Nom : Léonardo...</text>
6.   <text id="Descriptionréalisateur" >Nom : James...</text>
7.   <text id="NomVille" ¿Grenoble¡/text>
8.   <text id="TitreFilm" ¿Trois étoiles¡/text>
9.   <move id="BandeAnnonce" filename="TitreFilm" >
10.  <move id="InterviewActeur" filename="TitreFilm" >
11.  <move id="InterviewRéalisateur" filename="TitreFilm" >
12.  <audio id="MusicFond" filename="TitreFilm" >
13.  </media>
14.  <temporel>
15.  <composite id="scénario">
16.   <composite id="ParBande" duration="min :60 pref :90 max :150">
17.    <interval id="IntervalDesc"duration="min :60 pref :90 max :150"
media="DescriptionFilm"/>
18.    <interval id="IntervalBande" duration="min :60 pref :90 max :150"
media=" BandeAnnonce"/>
19.    <interval id="IntervalMusic" duration="min :60 pref :90 max :150"
media="MusicFond"/>
20.    <relations>
21.     <equals interval1="IntervalDesc" interval1="IntervalBande"/>
22.     <equals interval1="IntervalBande" interval1=" IntervalMusic"/>
23.    </relations>
24.   </composite>
```

```
25.    <composite id="ParAct" duration="min :40 pref :60 max :75">
26.     <interval id="IntervalDescAct" duration="min :40 pref :60 max :75"
media="DescriptionActeur"/>
27.     <interval id="IntervalAct" duration="min :40 pref :60 max :75" media=-
"InterviewActeur"/>
28.     <relations>
29.      <equals interval1="IntervalDescAct" interval1="IntervalAct"/>
30.     </relations>
31.    </composite>
32.    <composite id="ParRea" duration="min :40 pref :60 max :75">
33.     <interval id="IntervalDescRea" duration="min :40 pref :60 max :75"
media=" Descriptionréalisateur"/>
34.     <interval id="IntervalRea" duration="min :40 pref :60 max :75" media=-
"InterviewRéalisateur"/>
35.     <relations>
36.      <equals interval1="IntervalDescRea" interval1="IntervalRea"/>
37.     </relations>
38.    </composite>
39.    <relation>
40.     <meets interval1="ParBande" interval1="ParAct"/>
41.     <meets interval1="ParAct" interval1="ParRea"/>
42.    </relation>
43.   </composite>
44.   <composite>
45.    <interval id="IntervalVille" duration="min :140 pref :230 max :300"
media="NomVille"/>
46.    <interval id="IntervalTitre"duration="min :140 pref :230 max :300"
media="TitreFilm"/>
47.    <relations>
48.     <equals interval1="IntervalVille" interval1="IntervalTitre"/>
49.     <equals interval1="IntervalTitre" interval1="scénario"/>
50.    </relations>
51.   </composite>
52.  </temporel>
53.  <spatial>
54.   <composite>
55.    <region id="town" top="15" left="200" height="45" width="370"
media=" NomVille"/>
56.    <region id="title" top="392" left="200" height="35" width="370"
media="TitreFilm"/>
```

```
57.    <region id="video" top="80" left="20" height="292" width="350"
       media="BandeAnnonce"/>
58.    <region id="video" top="80" left="20" height="292" width="350"
       media ="InterviewActeur"/>
59.    <region id="video" top="80" left="20" height="292" width="350"
       media="InterviewRéalisateur"/>
60.    <region id="description" top="80" left="390" height="292" width-
       ="385" media ="DescriptionFilm"/>
61.    <region id="description" top="80" left="390" height="292" width-
       ="385" media=" DescriptionActeur"/>
62.    <region id="description" top="80" left="390" height="292" width-
       ="385" media =" Descriptionréalisateur"/>
63.   </composite>
64.  </spatial>
65. </madeus>
```

Annexe 2

Grammaire du langage XEF

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--     XML document type definition for XEF Language    -->


<!ELEMENT xef (medias,elements,relations?,priorites?)?>
<!ATTLIST xef %id; %spatial; %temporel;>

<!ELEMENT medias (texte|video|audio|image)*>
<!ATTLIST medias
   base CDATA \#IMPLIED>

<!ELEMENT texte (\#PCDATA)>
<!ATTLIST texte %id; %src;>

<!ELEMENT video EMPTY>
<!ATTLIST video %id; %src;>

<!ELEMENT audio EMPTY>
<!ATTLIST audio %id; %src;>

<!ELEMENT image EMPTY>
<!ATTLIST image %id; %src;>

<!ELEMENT elements (elem)*>

<!ELEMENT elem EMPTY>
<!ATTLIST elem %id; %media; %spatial; %temporel; %controle; >

<!ELEMENT refelem EMPTY>
<!ATTLIST refelem %ref;>

<!ELEMENT relations
     (refelem|relation|groupe|seq_spatial|seq_temporel|foreach)*>
```

```
<!ELEMENT relation (refelem,refelem)>
<!ATTLIST relation
    %id;
    type (meets|before|after|starts|ends|overlaps|during|equals|
          centerV|centerH|alignB|alignT|alignL|alignR) #REQUIRED;
    param CDATA \#IMPLIED; >

<!ELEMENT groupe
   (refelem|relation|groupe|seq_spatial|seq_temporel|foreach)*>
<!ATTLIST groupe %id; %spatial; %temporel; %controle; >

<!ELEMENT seq_spatial (refelem|groupe|seq_spatial|seq_temporel|foreach)*>
<!ATTLIST seq_spatial %id; %spatial; %temporel; %controle; >

<!ELEMENT seq_temporel (refelem|groupe|seq_spatial|seq_temporel|foreach)*>
<!ATTLIST seq_spatial %id; %spatial; %temporel; %controle; >

<!ELEMENT foreach
   (variable*,(refelem|relation|groupe|seq_spatial|seq_temporel|foreach)*)>
<!ATTLIST foreach %id; select PCDATA #REQUIRED; >

<!ELEMENT variable EMPTY>
<!ATTLIST variable %id; select PCDATA #REQUIRED; >

<!ELEMENT priorites (priorite)*>

<!ELEMENT priorite (priorite|refelem)*>
<!ATTLIST priorite
    %id;
    ordre (croissant|decroissant) \#IMPLIED;
    valeur CDATA \#IMPLIED >


<!ENTITY % id           "id  ID #IMPLIED" >
<!ENTITY % src          "src CDATA #IMPLIED" >
<!ENTITY % ref          "ref IDREF #IMPLIED" >
<!ENTITY % select       "select CDATA #IMPLIED" >
<!ENTITY % duration     "duration CDATA #IMPLIED" >
<!ENTITY % debut        "debut CDATA #IMPLIED" >
<!ENTITY % fin          "fin CDATA #IMPLIED" >
<!ENTITY % x1           "x1 CDATA #IMPLIED" >
<!ENTITY % y1           "y1 CDATA #IMPLIED" >
<!ENTITY % x2           "x2 CDATA #IMPLIED" >
<!ENTITY % y2           "y2 CDATA #IMPLIED" >
<!ENTITY % largeur      "largeur CDATA #IMPLIED" >
<!ENTITY % hauteur      "hauteur CDATA #IMPLIED" >
<!ENTITY % media        "media CDATA #IMPLIED" >
<!ENTITY % alternative  "alternative (vrai|faux) #IMPLIED" >
```

```
<!ENTITY % equilibrage  "equilibrage CDATA #IMPLIED" >
<!ENTITY % repli        "repli CDATA #IMPLIED" >


<!ENTITY % spatial      "%x1;%y1;%x2;%y2;%largeur;%hauteur;">
<!ENTITY % temporel     "%duration;%debut;%fin;">
<!ENTITY % controle     "%equilibrage;%repli;%alternative;">
```

Bibliographie

[All91] J.F. Allen, *Time and Time Again : The Many Ways to Represent Time*, International Journal of Intelligent Systems, vol. 6, , pp. 341-355, 1991.

[AG94] R. Allen, D. Garlan, *Formal connectors*, Technical Report CMU-CS-94-105, Carnegie Melon University, 1994.

[AWY93] G. Agha, P. Wegner, A. Yonezawa, *Research Directions in Concurrent Object-Oriented Programming*, MIT Press, 1993.

[App96] Apple Press, *OpenDoc Programmer's Guide*, Addison-Wesley Editions, 1996.

[BAT01] L. Bergmans, M. Aksit, B. Tekinerdogan, *Aspect Composition Using Composition Filters, in Software Architectures and Component Technology : The State of the Art in Research and Practice*, Edition : M. Aksit, Kluwer Academic Publishers, pp. 357 - 382, octobre 2001.

[BBB01] N. Belloir, J.M. Bruel, F. Barbier, *Formalisation de la relation Tout-Partie : application à l'assemblage des composants logiciels*, JC2001, Besançon, octobre 2001.

[BBL76] B.W. Boehm, J.R. Brown, M. Lipow, *Quantitative evaluation of software quality*, in 2nd International Conference on Software Engineering (IEEE), pp. 286-299, 1976.

[BCFFRS02] S. Boag, D. Chamberlin, M.F. Fernandez, D. Florescu, J. Robie, J. Siméon, *XQuery 1.0 : An XML Query Language*, W3C Working Draft, http ://www.w3.org/TR/xquery, 15 novembre 2002.

[BHL99] T. Bray, D. Hollander, A. Layman, *Namespaces in XML*, W3C Recommandation, http ://www.w3.org/TR/REC-xml-names, 14 janvier 1999.

[Bir01] D. Birngruber, *A Software Composition Language and Its Implementation*, In : Bjorner Dines, Broy Manfred, Zamulin Alexandre V. (Eds.) : Perspectives of System Informatics (PSI 2001), pp. 519-529, 2001.

[Bon] *GNOME Office*, http ://www.gnome.org/gnome-office/bonobo.shtml.

[Bon98] S. Bonhomme, *Transformation de documents structurés : une combinaison des approches déclaratives et automatiques*, Thèse en informatique, Université Joseph Fourier, décembre 1998.

[Bos97] J. Bosch. *Adapting Object-Oriented Components*, Proceedings of the 2nd Workshop on Component-Oriented Programming, 1997.

[BPMM00] T. Bray, J. Paoli, C.M.S. McQueen, E. Maler, *eXtensible Markup Language (XML) 1.0 (Second edition)*, W3C recommandation, 6 octobre 2000.

[BR02] F. Bes, C. Roisin, *Quels langages pour mieux contrôler le formatage des documents multimédia ?*, 5e Colloque International sur le Document Electronique (CIDE'5), Hammamet, Tunisie, 20-23 octobre 2002.

[BR02+] F. Bes, C. Roisin, *A Presentation Language for Controlling the Formatting Process in Multimedia Presentations*, Proceedings of the 2002 ACM Symposium on Document Engineering DocEng'2002, ACM, pp. 2-9, Virginia, USA, 8-9 novembre, 2002.

[BW98] .W. Brown , K. C. Wallnau, *The current state of CBSE*, IEEE Software, Vol. 15, n°5, pp. 37-46, septembre/octobre 1998.

[Car00] E. Cariou, *Spécification de composants de communication en UML*, Objets, Composants, Modèles (OCM'2000), Nantes, mai 2000.

[Car97] L. Carcone, *Formatage spatial dans un environnement d'èdition/présentation de documents multimédia*, Mémoire d'ingénieur CNAM, octobre 1997.

[CC88] J. Carlier, P. Chretienne, *Problèmes d'ordonnancement*, Editions : Masson, 1988.

[CD94] S. Cook, J. Daniels, *Designing object systems -Object-Oriented Modelling with syntropy*, Englewood Cliffs, New Jersey, Prentice Hall, 1994.

[CD99] J. Clark, S. DeRose, *XML Path language (XPath)*, W3C Recommandation, 16 novembre 1999, http ://xmlfr.org/w3c/TR/xpath.

[CH88] R. Chaffin, D.J. Hermann, *The nature of semantic relations : a comparaison of two approaches*, In : Evens, Martha W. (ed). Relational Models of the Lexicon, Cambridge University Press, pp. 289-334, 1988.

[CIMP01] D. Carlisle, P. Ion, R. Miner, N. Poppelier, *Mathematical Markup Language (MathML) Version 2.0*, W3C recommandation, http ://www.w3.org/TR/MathML2, 21 février 2001.

[Cla99] J. Clark, *XSL Transformation (XSLT) version 1.0*, W3C Recommandation, 16 novembre 1999, http ://www.w3.org/TR/xslt.

[Cla99+] J. Clark, *Associating style sheets with xml documents, Version 1.0*, W3C recommandation, 29 juin 1999, http ://www.w3.org/TR/xml-stylesheet.

[CLBBD01] T. Coupaye, R. Lenglet, M. Beauvois, E. Bruneton, P. Déchamboux, *Composants et composition dans l'architecture des systèmes répartis*, Journées Composants 2001, Besançon, 2001.

[COM99] *Microsoft COM technologies*, http ://www.microsoft.com/com/tech/com.asp, 1999.

[COR02] *CORBA, CommonObject Request Broker Architecture : Core Specification*, Version 3.0, OMG specification, décembre 2002.

[Cou96] B. Coulange, *Réutilisation du logiciel*, Masson, Paris, 1996.

[CP90] J. Carlier, E. Pinson, *A practical use of Jackson's preemptive schedule for solving the job-shop problem*, Annals of Operations Research 26, pp. 269-287, 1990.

[DC97] *Dublin Core, The Dublin Core Metadata Initiative*, http ://www.dublincore.org, 1997.

[DFK01] A. Diaz, J. Ferraiolo, S. Kulseth, P.L. Hégaret, C. Lilley, C. McCathie-Nevile, T. Roy, R. Whitmer, *Component Extension (CX) API requirements*, Version 1.0, W3C.

[DKMR02] M. Dubinko, L.L. Klotz, R. Merrick, T. V. Raman, *XForms 1.0*, W3C Candidate Recommendation, http ://www.w3.org/TR/xforms, 12 novembre 2002.

[DR97] S. Ducasse, T. Richner, *Executable Connectors : Towards Reusable Design Elements*, Proceedings of the Sixth European Software Engineering Conference (ESEC/FSE 97), pp. 483-499, 1997.

[Duc02] F. Duclos, *Environnement de gestion de services non-fonctionnels dans les apllications à composants*, Thèse en informatique, Université de Joseph Fourier, octobre 2002.

[Dur99] J-Y. V-Dury, *Circus : un générateur de composants pour le traitement des langages visuels et textuels*, Thèse en informatique, Université de Joseph Fourier, juin 1999.

[DYK01] L.G. DeMichiel, L. Yalçinalp, S. Krishnan, *Enterprise JavaBeans^{TM} Specification*, Version 2, SUN Microsystems, Août 2001.

[EKS94] J. Eder, G. Kappel, M. Schreff, *Coupling and cohesion in object-oriented systems*, Technical Report, University of Klagenfurt, Autriche, 1994.

[FB96] N. Freed, N. Borenstein, *Multipurpose Internet Mail Extensions (MIME)*, RFC 2046, novembre 1996.

[FGS02] A. Farias, Y.G. Guéhéneuc, M. Südholt, *Integrating Behavio Protocols in Enterprise Java Beans*, OOPSLA, Workshop on Behavioral Semantics 2002, Washington, USA, 2002.

[GCLW01] D. Glazman ,T. Celik, P. Linss, J. Williams, *Selectors*, W3C Candidate Recommendation, http ://www.w3.org/TR/css3-selectors, 13 novembre 2001.

[GHJV95] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns : Element of reu-sable Object-Oriented Software*, Addison-Wesley, 1995.

[GJ79] . R. Garey, D. S. Johnson, *Computers and Intractability : A Guide to the Theory of NP-Completeness*, Edition : W. H. Freeman and company, San Francisco, 1979.

[GMMW03] P. Grosso, E. Maler, J. Marsh, N. Walsh, *XPointer Framework*, W3C recommandation, http ://www.w3.org/TR/2003/REC-xptr-framework-20030325, 25 mars 2003.

[GNO] *GNOME Office*, http ://www.gnome.org/gnome-office/index.shtml.

[GPV94] N. Guarino, S. Pribbenow, L. Vieu, *Parts and wholes : conceptual parts-whole relations and formal mereology*, ECAI'94 workshop, Amsterdam, 1994.

[Hei95] S.Heiler, *Semantic interoperability*, ACM Computing Surveys, vol 27, n°2, pp. 271-273, 1995.

[HGP92] M. Halper, J. Geller, Y. Perl, *Part relations for object oriented databases,* Proceeding of the 11th international conference on the entity relationship approach, Karlsruhe, pp. 406-422, 1992.

[HP02] H. Hosoya, B.C. Pierce, *XDuce : A typed XML processing language*, ACM Transactions on Internet Technology, 2002.

[IETF98] T. Berners-Lee, R. Fielding, L. Masinter, *Internet Engineering Task Force RFC 2396 : Uniform Resource Identifiers (URI) : Generic Syntax*, 1998.

[ISO95] ISO, *Reference Model of Open Distributed Processing*, International Standard ISO/IEC 10746, ITU/T Recommendations X.901 :904, 1995.

[JO95] Z. Jin, A.J. Offutt, *Integration testing based on software couplings*, Tenth Annual Conference on Computer Assurance (COMPASS 94), IEEE Computer Society Press, pp. 13-23, Gaithersburg MD, juin 1995.

[Kan76] A.H.G.R. Kan, *Machine scheduling problems : classification, complexity and computation*, Nijhoff, The Hague, 1976.

[Kic96] G. Kiczales, *Beyond the black box : open implementation*, IEEE Software, Vol. 13, n°1, 1996.

[KKL00] H. Kabaili, R. Keller, F. Lustman, *Class cohesion as predictor of changeability : An empirical study*, In Proceedings of the Fourth International Workshop on Quantitative Approaches in Object-Oriented Software Engineering, pp. 29-38, Sophia Antipolis, juin 2000.

[KL91] H. Kautz, P.B. Ladkin, *Integrating metric and qualitative temporal reasoning*, In Proceedings of AAAI-91, pp. 241-246, 1991.

[KLMMLLI97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C.V. Lopes, J-M. Loingtier, J. Irwin, *Aspect-Oriented Programming*, Proceedings European Conference on Object-Oriented Programming, 1997.

[Lay97] N. Layaïda, *Madeus : système d'édition et de présentation de documents structurés multimédia*, Thèse en informatique, Université de Joseph Fourier, juin 1997.

[Led03] Y. Ledru, *Architecture logicielle : cours de génie logicielle pour RICM*, Université Joseph Fourier, Grenoble, 2003.

[Lee99] M. Lee, *Coupling and cohesion*, GMU graduate students of SWSE 619, George Mason University, 1999.

[Lew98] S.M. Lewandowski, *Frameworks for component-based client/serveur computing*, ACM Computing Surveys, vol. 30, n°1, pp. 3-27, mars 1998.

[Lex] *Lexique du projet de traduction de Dubian*, http ://www.debian.org/international/french/aide.fr.html.

[LGH02] S. Little, J. Geurts, J. Hunter, *Dynamic Generation of Intelligent Multimedia Presentations through Semantic Inferencing*, 6th European Conference on Research and Advanced Technology for Digital Libraries, pp. 158-189, septembre 2002.

[LimSee1] LimSee : Un éditeur temporel pour les documents au format SMIL, http ://opera.inrialpes.fr/LimSee.html

[Lin97] G. Lindén, *Structured document transformations*, Thèse en informatique, Department of computer science, University of Helsinki, 1997.

[LLWNRCB00] A. Le Hors, P. Le Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, S. Byrne, *Document Object Model (DOM)*, W3C Recommandation, http ://www.w3.org/TR/DOM-Level-2-Core, 13 novembre 2000.

[Meg01] D. Megginson, *SAX : Simple API for XML*, http ://www.saxproject.org, 2001.

[Mey88] B. Meyer, *Object-Oriented software construction*, Prentice-Hall Editions, 1988.

[Mey97] B. Meyer, *Object-oriented software construction*, Prentice Hall, 1997.

[MK89] M. Murata, K. Kusumoto, *Daemon : another way of invoking methods*, Journal of object oriented programming 2(2), 1989.

[MN98] T.D. Meijler, O. Nierstrasz, *Cooperative information systems - Trends and directions*, chap. Beyond Objects : Components, Academics Press, pp. 49-78, San Diego, 1998.

[Moc87] P. Mockapetris, *Domain Names - Impementation and specification, RFC1035*, http ://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1035.html, novembre 1987.

[MP00] R. Marvie, M.C. Pellegrini, *Etat de l'art des modèles de composants*, Technical Report 2 of the CESURE RNRT Project, mai 2000.

[MT00] N. Medvidovic, R.N. Taylor, *A classification and comparaison framework for software architecture description languages*, IEEE Transactions on Software Engineering, vol. 26, n°1, pp. 70-93, janvier 2000.

[Mye74] G. Myers, *Reliable software throught composite design*, Mason and Lipscomb Editions, New York, 1974.

[Nap92] A. Napoli, *Représentations à objets et raisonnement par classification en intelligence artificielle*, Thèse en informatique, S.T.M.I.A, Univerité de Nancy, janvier 1992.

[OHBS94] H. Ossher, W. Harrison, F. Budinsky, I. Simmonds, *Subject-Oriented Programming : Supporting Decentralized Development of Objects*, Proceedings of the 7th IBM Conference on Object-Oriented Technology, 1994.

[OHK93] A.J. Offutt, M.J. Harrold, P. Kolte, *A software metric system for module coupling*, The journal of systems and software, pp. 295-308, 1993.

[Ous97] C. Oussalah, *Ingénierie objet, concepts et techniques*, Masson, Paris, 1997.

[Pin95] M. Pinedo, *Scheduling, theory, algorithms, and systems*, Prentice Hall, Englewood Cliffs, 1995.

[PJ80] M. Page-Jones, *The pratical guide to structured systems design*, Yourdon press, New York, 1980.

[PV02] F. Plasil, S. Visnovsky, *Behavior Protocols for Software Components*, IEEE Transactions on Software Engineering, IEEE, janvier 2002.

[RHA95] H. Richy, Ch. Hérault, J. André, *Notion de feuilles de style (résumé étendu)*,Cahiers GUTenberg, 21, pp. 127-134, juin 1995.

[Riv99] D. Rivreau, *Problèmes d'Ordonnancement Disjonctifs : Règles d'élimination et Bornes Inférieures*, Thèse en informatique, Université de Technologie de Compiègne, December 1999.

[RLJ99] D. Raggett, A. Le Hors, I. Jacobs, *HTML 4.01 Specification*, W3C recommandation, http ://www.w3.org/TR/html4, 24 décembre 1999.

[Rog97] D. Rogerson, *Inside COM, Microsoft programming series*, Microsoft Press, 1997.

[RV] C. Roisin, L. Villard, *Transformation de documents dans les présentations multimédia*,

[Ryo01] J.W. Ryon, *Cours software engineering CIS 490*, New Jersey Institute of technology (NJIT), 2001.

[SDKRYZ95] M. Shaw, R. Deline, D.V. Klein, T.L. Ross, D.M. Young, G. Zelznik, *Abstractions for Software Architectures and Tools to Support Them.* IEEE Transactions on Software Engineering, avril 1995.

[SG95] M.Shaw, D. Garlan, *Software architecture : Perspectives on an emerging discipline*, Prentice Hall, avril 1995.

[SGML86] International Standard ISO 8879 Information Processing, *Text and Office Systems-Standardized Generalized Markup Language (SGML)*, UDC 681.3.06 Ref. No. ISO 8879-1986(E), First Edition 15-10-1986.

[Sha95] M. Shaw, *Architectural issues in software reuse : It's not just the functionnality, it's the packaging*, Proceedings of the IEEE Symposium on Software Reuse, pp. 3-6, 1995.

[SJ] M.M. Schrage, J. Jeuring, *Xprez : A Declarative Presentation Language for XML*, Utrecht University.

[SLMD96] P. Steyaert, C. Lucas, K. Mens, T. D'Hondt, *Reuse contracts : managing the evolution of reusable assets*, Proceeding of OOPSLA'96, pp. 268-285, 1996.

[SMC74] W. Stevens, G. Myers, L. Constantine, *Structured design*, IBM system journal, Vol. 13, pp. 115-139, 1974.

[SMIL2] J. Ayars, D. Bulterman, A. Cohen, K. Day, E. Hodge, P. Hoschka, E. Hyche, M. Jourdan, M. Kim, K. Kubota, R. Lanphier, N. Layaïda, T. Michel, D. Newman, J.V. Ossenbruggen, L. Rutledge, B. Saccocio, P. Schmitz, W.T. Kate, *Synchronized Multimedia Integration Language (SMIL 2.0)*, W3C Recommandation, http ://www.w3.org/TR/smil20, 2001.

[SN99] J.G. Schneider, O. Nierstrasz, *Software architectures - advances and applications, chap.Components, Scripts and Glue*, L. Barroca, J.Hall, and P.Hall Editions, pp. 13-25, 1999.

[Sol99] J.J. Solari, *Recommandation XML-Namespace du W3C en version française*, http ://www.yoyodesign.org/doc/w3c/xml-namespace/Overview.html#URIRef, 1999.

[SP96] C. Szyperski, C. Pfister, *WCOP'96 Workshop Report, Workshop Reader ECOOP'96* , pp. 127- 130, juin 1996.

[SVG1] J. Bowler, others, *Scalable Vector Graphics (SVG) 1.0 Specification*, W3C Recommendation, http ://www.w3.org/TR/SVG, 04 septembre 2001.

[Szy97] C. Szyperski, *Component Software - Beyond Object-Oriented Programming*, New York, Addison-Wesley, 1997.

[Tar00] L. Tardif, *Kaomi : réalisation d'une boîte à outils pour la construction d'environnements d'édition de documents multimédias*, thèse de doctorat, Institut National Polytechnique de Grenoble, décembre 2000.

[TBMM01] H.S. Thompson, D. Beech, M. Maloney, N.Mendelsohn, *XML Schema Part 1 : structures* , W3C Recommandation, http ://www.w3.org/TR/xmlschema-1, 02 mai 2001.

[Thu03] T.T. Thuong, *Modélisation et traitement du contenu des médias pour l'édition et la présentation de documents multimédias*, Thèse en informatique, Institut national polytechnique de Grenoble, février 2003.

[TV01] J.M. Troya, A. Vallecillo, *Controllers : reusable wrappers to adapt software components*, Information and Sofware Technology, vol. 43, n° 1, pp. 189-202, 2001.

[Vil02] L. Villard, *Modèles de documents pour l'édition et l'adaptation de présentations multimédias*, Thèse en informatique, Institut national polytechnique de Grenoble, mars 2002.

[Wat90] D. Watt, *Programming language concepts and paradigms*, International series in computer science, PrenticeHall, 1990.

[WCH87] M.E. Winston, R. Chaffin, D.J. Hermann, *A taxonomy of part-whole relations*, Cognitive Science, Vol. 11, pp. 417-444, 1987.

[Wyd01] B. Wydaeghe, *PACOSUITE, Component Composition Based on Composition Patterns and Usage Scenarios*, PhD thesis, Vrije Universiteit Brussels, 2001.

[XHTML1] S. Pemberton, others, *XHTML 1.0 : The Extensible HyperText Markup Language (Second Edition)*, A Reformulation of HTML 4 in XML 1.0, W3C Recommendation, revised 1 August 2002, http ://www.w3.org/TR/xhtml1, 26 janvier 2000.

[XMLS] *XML Script*, http ://www.xmlscript.org, 11 Décembre 2001,

[XSL1] S. Adler, A. Berglund, J. Caruso, S. Deach, T. Graham, P. Grosso, E. Gutentag, A. Milowski, S. Parnell, J. Richman, S. Zilles, *extensible Stylesheet Language (XSL) Version 1.0*, W3C recommandation, http ://www.w3.org/TR/xsl, 15 octobre 2001.

[XSmiles] *X-Smiles, An open XML-browser for exotic devices*, http ://www.xsmiles.org.

[Zel00] M.V. Zelkowitz, *Module design issues*, MSWE 607 Software Life Cycle Methods and Techniques (UMCP), University of Maryland, 2000.