INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque

THÈSE

pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : « Informatique : systèmes et communication »

préparée au laboratoire de l'Institut National de Recherche en Informatique et en Automatique

dans le cadre de **l'Ecole Doctorale « Mathématiques, Sciences et technologies de l'Information »**

présentée et soutenue publiquement

par

Lionel Villard

le 21 Mars 2002

Titre :

Modèles de documents pour l'édition et l'adaptation de présentations multimédias

Directeur de thèse :

Cécile Roisin

JURY

M[r] Jacques Mossière	, Président
M[r] Marc Nanard	, Rapporteur
M[r] Kristoffer H. Rose	, Rapporteur
M[me] Cécile Roisin	, Directeur de thèse
M[r] Nabil Layaïda	, Co-encadrant
M[me] Joëlle Coutaz	, Examinatrice
M[r] Franck Duluc	, Examinateur

REMERCIEMENTS

Pour commencer, je tiens à remercier Vincent Quint et Cécile Roisin pour m'avoir accueilli dans le projet Opéra. Je remercie également Nabil Layaïda pour son chaleureux soutien tout au long de cette thèse.

Je remercie également tous les membres du jury :

Jacques Mossière professeur à l'INPG de m'avoir fait l'honneur de présider mon jury de thèse,

Marc Nanard professeur au CNAM et Kristoffer H. Rose chargé de recherche chez IBM pour avoir jugé mon travail,

Joëlle Coutaz professeur à l'UJF et responsable du projet IIHM avec qui j'ai fait mes tous premiers pas dans le monde de la recherche,

et Franck Duluc ingénieur chez Airbus France qui m'a initié à l'autre monde qu'est l'industrie à travers le domaine particulier de l'aéronautique.

Je remercie tout particulièrement Cécile Roisin et Nabil Layaïda pour m'avoir tous deux encadrés durant ces années et pour leurs nombreuses relectures du mémoire.

Je tiens à remercier également tous les membres du projet Opéra et W3C, permanents ou non, anciens et nouveaux, à savoir Tien Tran_Thuong contributeur important de Madeus 2.0, Muriel Jourdan, Frederic Bes, Frederic Séraphine, Laurent Carcone, Laurent Garçon, Laurent Tardif, Tayeb Lemlouma, Vincent Kober, Julien Guyard, Stephane Bonhomme, Loay Sabry, mes stagiaires de Maîtrise Nicolas Seytre et Stephane Bissol, mes stagiaires de DESS Véronique Zonca et Michael Tissot et tous les autres.

Enfin et surtout, je remercie toute ma famille, mes parents qui, sans eux, je ne serais pas là, mon frère et ma (future) belle sœur, ma sœur et son (futur) mari pour avoir mis au monde le plus beau des joyaux, j'ai nommé Manon, sans oublier tous les autres. Je remercie également tous mes ami(e)s Ka, Elo, Marie, Zahia, Mag, Isa, …

Pour finir, je remercie tous particuliérement l'élue du mon cœur Nadia, pour m'avoir accompagné et soutenu tout au long de mes études.

TABLE DES MATIERES

TABLE DES FIGURES

Chapitre 1

Introduction

1. Contexte

Ces dernières années ont vu l'apparition d'une multitude de nouveaux moyens permettant à un lecteur d'accéder et de consulter l'information qui l'intéresse. De nos jours, l'accès à l'information s'effectue de façon très hétérogène, que ce soit au moyen d'un accès local ou encore d'un accès à travers le réseau. Avec l'avènement du World Wide Web, ce n'est plus un groupe restreint de personnes qui consulte et produit l'information mais potentiellement les utilisateurs du monde entier dans toute leur diversité. Plus récemment, ce sont les supports physiques à partir desquels l'information est consultée qui ont fait leur révolution. Désormais on peut consulter les informations boursières sur son téléphone cellulaire, gérer son budget avec un assistant personnel (PDA), et bientôt faire ses courses à partir de son réfrigérateur. Le défi des systèmes traitant l'information est de faire face à cette diversité croissante des modes d'accès, qui constituent le profil ou *contexte utilisateur*, en proposant des solutions qui répondre au besoin de chacun de ces contextes.

En parallèle, le contenu de l'information a lui aussi évolué. Il est devenu de plus en plus riche en intégrant des contenus complexes comme la vidéo, des dessins vectoriels ou encore les animations tridimensionnelles. Ce contenu est regroupé dans ce qu'on appelle désormais des *documents multimédias*. Ces documents organisent à la fois dans l'espace et dans le temps les différents contenus qui les composent, et permettent d'éventuelles interactions avec le lecteur. L'intégration de toutes ces dimensions dans la structure globale d'un document nécessite une nouvelle approche pour éditer et restituer des documents multimédias appelée *édition et présentation multimédia*.

La diffusion du multimédia est effective de nos jours comme le témoigne les nombreuses utilisations à travers des domaines d'application aussi variés que l'éducation à distance, la médecine, les jeux, le tourisme ou l'aéronautique. Cependant, cette diffusion s'est faite sans une évolution des systèmes d'accès à l'information. En effet, ces derniers ne répondent pas à ce besoin de concevoir des documents multimédias qui s'adaptent à leur domaine d'utilisation. L'approche de conception présente dans les outils d'édition multimédia actuels [DoCoMo01, LimSee, Director 8.5, GRiNS] est insuffisante car trop spécifique. Pour limiter les coûts de réalisation en factorisant au mieux les tâches de conception de documents multimédias, il est nécessaire d'offrir le moyen de spécifier les documents à un niveau *générique* à travers la notion de *classe de documents* multimédias. De plus, les systèmes existants ne prennent en compte ni la variété des supports pour visualiser les présentations multimédias ni les capacités du lecteur.

Le travail de cette thèse a pour but de contribuer aux thèmes d'édition et de présentation de documents multimédias qui appartiennent à une classe et qui soient adaptables au contexte utilisateur.

2. Cadre de travail

Ce travail de thèse s'est déroulé au sein du projet Opéra à l'INRIA Rhône-Alpes. Il a été mené dans le contexte d'une collaboration avec la société Airbus France[1].

La société Airbus France s'intéresse à la production d'avions civils et militaires. Notamment, elle assure la maîtrise d'œuvre de l'intégration de la documentation technique pour les gammes des avions qu'elle produit. Cette documentation décrit l'avion lui-même et son fonctionnement. Elle présente les nombreuses spécificités suivantes [François 97] :

- une grande diversité de manuels techniques. La documentation technique est décomposée en trois familles selon l'usage du manuel : la formation des équipages grâce à la documentation d'instruction, l'usage des avions grâce à la documentation opérationnelle, et l'entretien des appareils grâce à la documentation d'entretien ;
- une documentation très volumineuse. Pour donner un exemple, un Airbus A320 est livré avec 39 manuels différents ce qui représente environ 800 000 pages et trois tonnes de papier ;
- une documentation à base de texte et d'illustrations ;
- une documentation personnalisée pour chaque avion livré. En effet, les avions A320 délivrés à Air France sont différents de ceux délivrés à Air Canada. Deux documentations techniques distinctes doivent donc être produites. De plus, au sein d'une même compagnie aérienne, les avions de même type ne sont pas strictement identiques ;
- une documentation soumise à des règles strictes. La documentation pour les avions civils est normalisée par l'ATA. Le cycle de révision de la documentation est de trois mois. Enfin, la durée de vie de la documentation doit être la même que celle d'un avion, c'est-à-dire plusieurs dizaines d'années.

Les progrès technologiques en matière de documents multimédias et d'accès à l'information ont conduit Airbus France à s'intéresser à ces technologies pour la production et la diffusion de manuels multimédias. Le résultat de cet intérêt se reflète en particulier par la publication d'une thèse sur la modélisation d'un fond documentaire multimédia [Duluc 00a]. À la suite de ces travaux, plusieurs thèmes émergeants sont apparus, en particulier celui-ci qui porte sur la définition de classes de documents multimédias adaptées à une documentation aéronautique d'exploitation des avions. C'est dans cette perspective qu'une collaboration entre l'INRIA et Airbus France s'est instaurée et qui se concrétise par ce mémoire de thèse.

Le projet Opéra s'intéresse aux documents électroniques : documents structurés, hypertextes et multimédias. Il étudie les modèles de documents qui rendent compte à la fois de leur organisation logique ou abstraite, de leur présentation graphique, de leur contenu et de leur aspect temporel. Il met également au point des techniques d'édition et de présentation qui s'appuie sur ces modèles. Récemment, le projet Opéra s'est intéressé au besoin d'adapter la présentation d'un document multimédia au contexte de l'utilisateur, c'est-à-dire le terminal qu'il utilise, l'environnement qui l'entoure et ses propres capacités. Les travaux contenus dans

[1] Airbus France est une filiale du groupe EADS (European Aerospace Defence and Space compagny)

cette thèse sont en adéquation avec les thèmes de recherche du projet Opéra puisqu'ils traitent de modèles structurés pour la présentation multimédia adaptée au contexte utilisateur. De plus, un chapitre entier est consacré aux techniques d'édition.

Les actions de recherche du projet Opéra trouvent leur application dans plusieurs prototypes :

- Thot est un outil interactif pour l'édition de documents structurés qui respectent des modèles. Cet outil a mis en place les principes d'édition et de présentation de documents respectant un modèle dans un cadre statique. Dans ce mémoire nous étendons ces principes aux documents dynamiques ;

- Byzance est un logiciel d'édition coopérative sur le Web qui adapte le document aux droits qu'un auteur possède sur un document. Dans cette thèse nous considérons l'adaptation dans un cadre plus large ;

- Madeus, un système d'édition et de présentation de documents multimédias reposant sur une approche spécifique. Nous proposerons un système d'édition similaire mais reposant sur une approche générique ;

- VideoMadeus, un environnement qui permet l'édition de la structure interne de média complexes comme la vidéo et l'intégration de fragments de vidéos ainsi structurés au sein de documents multimédia. Cet outil est développé dans le cadre d'une thèse qui est en cours. Le modèle de document étudié dans ce mémoire a été défini communément avec le modèle de document de VideoMadeus ;

- LimSee, un éditeur temporel pour les documents multimédias spécifiés selon le standard SMIL 1.0 [SMIL 98].

Nos expérimentations s'intégreront dans le prototype Madeus et dans un logiciel de transformation de document externe au projet.

3. Motivation et objectifs

La conception de systèmes auteur multimédia est un grand défi pour les industriels comme l'Airbus France qui manipulent de grandes quantités d'informations. Il y a quelques années, la notion de classe de document a été introduite pour les documents statiques (cf. SGML [ISO 86]) pour augmenter la productivité, la qualité et la pérennité des documents. Cette notion repose à la fois sur la définition formelle d'une classe de document en utilisant un langage de définition de type de document (DTD) et sur la séparation des informations de contenu des informations de présentation. Cette séparation suppose l'utilisation d'un traitement réunissant ces deux ensembles d'informations pour produire la présentation finale restituée au lecteur. Pour les documents statiques, ce traitement repose sur l'application de feuilles de styles constituées d'une liste de règles associant un fragment du contenu à un style graphique (couleur, police de caractères, mise en page, etc.) de présentation. L'application de ces règles suit l'organisation du contenu et donc les présentations produites sont très proches de cette organisation, ce qui est souvent le cas pour les documents statiques : l'ordre de définition, par exemple, d'un ensemble de paragraphes correspond à l'ordre conventionnel d'affichage de ces paragraphes, c'est-à-dire de haut en bas. En ce qui concerne les documents multimédias, une telle approche est insuffisante pour plusieurs raisons. D'une part, la notion de styles graphiques n'est pas suffisante car elle ne prend pas en compte les besoins d'organisation

spatiale et temporelle d'un document multimédia. Et d'autre part, la structure du contenu peut être amenée à différer complètement de la structure des présentations multimédias. Ces limitations peuvent être évitées en offrant une modélisation des informations de présentation dans l'espace et le temps, et en permettant leur rattachement aux informations de contenu indépendamment de leurs structures. Ce dernier besoin nous a poussé à explorer les techniques de *transformation* de documents dont un des objectifs est de répondre à ce besoin.

En parallèle à ce besoin, un des freins au développement de la lecture des documents électroniques est lié à la difficulté de leur utilisation dans certaines conditions relatives à l'arrivée de nouveaux terminaux. Généralement le contenu du document est souvent trop important pour être lu à partir de supports ayant un faible espace d'affichage comme les PDA. En effet, il est souvent nécessaire de naviguer à la fois dans l'axe horizontal et dans l'axe vertical pour visualiser entièrement le document, ce qui est une tâche pénible. Pour éviter ces désagréments, il est nécessaire de produire en amont des présentations qui soient adaptées au contexte utilisateur. L'utilisation d'une approche fondée sur la séparation du contenu des informations de présentation en utilisant un processus de transformation est une piste naturelle pour produire des documents multi-cibles.

Les solutions mises en œuvre pour répondre aux deux besoins cités précédemment doivent être en adéquation avec le besoin primordial de l'*édition* de documents. Ce besoin est largement couvert par une multitude d'outils d'édition, que ce soit pour les documents statiques et, dans une moindre mesure, pour les documents multimédias. Cependant nous avons souligné dans la section 1 le manque de généricité de ces outils lorsqu'il s'agit de concevoir des documents multimédias. De plus, l'édition du document dans ces outils s'effectue généralement sur sa représentation finale. Or cette représentation n'est plus unique puisqu'il en existe autant que de contextes utilisateur. Aucun travail ne traite de ces deux problèmes qui sont, à notre sens, fondamentaux. Une des raisons essentielle est la difficulté de mettre en œuvre des outils d'édition réactifs. En effet, comme l'édition s'effectue sur la représentation finale, toute modification du contenu doit être répercutée au plus vite sur sa représentation en ne recalculant que les informations nécessaires. Or pour le moment il n'existe pas de solution efficace lorsque la représentation est produite par transformation, à cause de la complexité de ce traitement.

Dans cette thèse nous avons la volonté d'offrir des solutions pour faciliter la production de documents multimédias adaptés selon les critères de performance, de qualité et d'accessibilité par des auteurs non informaticiens. Nous serons donc amené à chercher des solutions de *transformations incrémentales* pour mettre à jour la représentation du contenu.

Ces motivations nous ont conduit à aborder cette thèse selon les objectifs suivants :

- choisir ou définir un modèle de présentation de documents multimédias pouvant être intégrés dans une approche de séparation du contenu de ses présentations ;
- choisir ou définir un langage de transformation permettant de produire des présentations multimédias à partir d'une source d'informations appartenant à une classe ;
- proposer un système qui permet de créer des présentations adaptées au contexte utilisateur. Un tel système est appelé *système adaptable* ;

- proposer un éditeur qui permet à la fois d'éditer le contenu et la spécification de la présentation ;
- concevoir des algorithmes pour mettre à jour le résultat d'une transformation de façon incrémentale.

Les objectifs présentés ci-dessus se situent clairement dans un contexte applicatif. Par conséquent, toute proposition théorique pour répondre à ces objectifs doit être validée à travers la réalisation de prototype. Deux types de résultats sont donc visés par cette thèse :

- des résultats théoriques porteront sur la spécification d'un modèle de documents multimédias génériques, la proposition d'une architecture d'un système adaptable, d'un modèle d'édition de transformation et d'algorithmes pour réaliser des transformations incrémentales ;
- des résultats pratiques constitués d'un système adaptable pour la présentation de documents multimédias appartenant à une classe et qui soit adaptée au contexte utilisateur, ainsi que d'un éditeur basé sur un processus de transformation incrémental permettant la conception de telles présentations.

4. Plan du mémoire

La suite de ce mémoire est organisée en deux parties. La première partie présente un état de l'art et la seconde partie décrit à la fois les aspects théoriques de notre contribution et les implantations qui en découlent. Ces parties sont les suivantes :

Partie I État de l'art : chapitres 2 et 3

Chapitre 2 Génie documentaire. Les travaux de cette thèse se situent clairement dans le domaine du génie documentaire qui positionne le document au cœur de toutes les préoccupations. L'objectif de ce second chapitre est de familiariser le lecteur à ce domaine en donnant les définitions fondamentales et en exposant les principaux besoins liés à ce domaine. Ensuite, nous étudions les travaux existants concernant une partie des besoins énoncés ci-dessus concernant la modélisation, la présentation et l'édition de documents statiques et multimédias. Nous complétons cette étude par les travaux relatifs à la conception d'algorithmes incrémentaux.

Chapitre 3 Adaptation de documents multimédias. Dans ce chapitre, nous étudions les travaux émergents concernant l'adaptation de documents multimédias au contexte utilisateur. Nous avons découpé cette étude en trois parties principales, la première portant sur les travaux concernant la modélisation du contexte utilisateur. Nous les décrivons et nous proposons une taxonomie des propriétés caractérisées par ces modèles. Ensuite nous effectuons un état de l'art sur les techniques traitant de l'une ou de plusieurs de ces propriétés. Enfin la troisième partie concerne l'étude des systèmes dédiés au besoin d'adaptation.

Partie II Contribution : chapitres 4 à 7

Chapitre 4 Un système de production de documents multimédias adaptés au contexte de l'utilisateur. Dans ce chapitre, nous décrivons notre proposition d'un système adaptable permettant de produire des documents multimédias adaptés au contexte de l'utilisateur. Nous décrivons son architecture et le rôle précis de chacun de ses composants. Pour chacun de ces

composants, nous identifions les langages et les techniques décrites dans le chapitre 3 permettant de réaliser l'adaptation, et nous montrons comment ces composants interagissent. En particulier, cette étude va nous conduire à choisir le langage de transformation XSLT défini par le W3C et le langage de description de présentation multimédia Madeus 2.0. Une partie de ce système a été validée à travers plusieurs expériences que nous relatons.

Chapitre 5 Outil d'aide pour l'édition de documents adaptables. L'objectif de ce chapitre est de proposer un outil pour faciliter la tâche de l'auteur pour éditer les documents adaptables. Cet outil repose sur une approche interactive : nous proposons donc d'identifier les actions possibles au niveau de l'interface utilisateur puis nous proposons des solutions de génération à appliquer sur les documents pour chacune de ces actions. Nous verrons que le cœur de cet outil est un processeur de transformation.

Chapitre 6 Transformation incrémentale. La réussite de l'outil d'aide décrit dans le chapitre précédent repose sur la mise en œuvre d'un processeur de transformation incrémental pour le langage XSLT. Pour montrer la pertinence des solutions que nous proposons, nous avons choisi une approche formelle pour modéliser la sémantique de ce langage. Grâce à cette modélisation formelle, nous serons en mesure d'analyser les conséquences d'une modification du document adaptable sur ses multiples représentations, et ainsi proposer des solutions théoriques pour répondre à ces conséquences. Nous confrontons ensuite ces solutions par la réalisation d'un processeur de transformation incrémental que nous évaluons en le comparant à un processeur non incrémental.

Chapitre 7 Conclusion. Dans ce dernier chapitre, nous effectuons un résumé sur l'apport de cette thèse. Nous proposons ensuite plusieurs perspectives dans ce nouveau thème de recherche concernant l'adaptation de documents multimédias.

PREMIERE PARTIE

ÉTAT DE L'ART

Chapitre 2

Génie documentaire

Le premier objectif de ce chapitre est de familiariser le lecteur au domaine du génie documentaire. Le second objectif est de positionner le contenu de ce mémoire par rapport aux problématiques posées dans ce domaine. Pour cela, nous donnons quelques définitions et exposons les principaux problèmes liés au génie documentaire. Puis nous effectuons un tour d'horizon des techniques relatives aux objectifs que nous nous sommes fixés dans l'introduction : la modélisation, la présentation et l'édition.

1. Définitions et problématiques

1.1. Définitions

Le sujet de ce mémoire de thèse se situe principalement dans le domaine du génie (ou ingénierie) documentaire. L'élément central de ce domaine est le *document*, par opposition au génie logiciel qui est centré sur les *traitements*. De façon générale, « *le génie documentaire s'intéresse aux principes, aux outils et aux processus permettant d'améliorer notre capacité à créer, à gérer et à maintenir d'importants ensembles de documents* » [Munson01]. Dans ce mémoire, nous nous focaliserons surtout sur les principes, outils et processus permettant de créer et de présenter des documents multimédias.

Le premier terme que nous définissons est le terme *document*. Ce terme provient étymologiquement du latin *documentum* signifiant « ce qui sert à instruire ». Cependant, à cause des évolutions culturelles, sociales et scientifiques, la racine étymologique ne suffit plus à caractériser la notion de document comme elle est perçue de nos jours. En particulier, avec l'avènement de l'informatique, de nouvelles technologies sont apparues comme support du document. Ces technologies sont caractérisées par des capacités et des contraintes différentes. Un document peut prendre alors plusieurs formes selon son contexte, c'est pourquoi les documentalistes s'entendent pour considérer une vue fonctionnelle de ce que constitue un document plutôt qu'une vue sur sa forme (papier, cassette, zoo[2], etc.) [Briet 51, Buckland 97, Schamber 96]. Pour limiter la discussion sur ce qu'est un document, nous avons choisi une définition plus pragmatique, issue du domaine informatique, qui insiste sur la notion de propriétés et d'usages du document [Quint 87, Roisin 99a] :

> *« Le document désigne un ensemble cohérent et fini, d'informations plus ou moins structurées, perceptibles, à usage défini et représenté sur un support concret »*

Un des points intéressants de cette définition est l'aspect perceptible du document. La plupart du temps, la perception d'un document s'effectue à travers une représentation graphique de celui-ci. Cependant, on peut noter une évolution dans ce domaine puisque, entre autres, la

[2] Suzanne Briet considère un animal dans un zoo comme étant un document. Le zoo est par conséquent une forme de document.

représentation sonore est de plus en plus utilisée pour percevoir le document. C'est déjà le cas lorsque le contenu du document est de nature sonore mais ce mode de perception se généralise à l'ensemble des contenus du document en particulier par l'usage de « navigateurs audio » [VoiceXML 01].

Les autres termes utilisés en génie documentaire héritent pour la plupart du vocabulaire employé dans le paradigme orienté objet. Les documents qui obéissent à un même ensemble de contraintes sont regroupés en familles appelées *modèles de document* (ou classes de document). Lorsqu'un document respecte un modèle de document, on dit que c'est une *instance* de ce modèle. La définition formelle d'un modèle de document s'effectue grâce à un *méta-modèle*. Le méta-modèle permet de définir des contraintes sur la classe de document.

Le stockage des documents nécessite une représentation physique du contenu du document. Cette représentation physique est appelée *format* de document.

Le dernier terme que nous définissons est le terme *application*. Dans notre contexte, une application dénote une utilisation du document. Initialement, le document était exclusivement présent dans les chaînes éditoriales. Les applications inclues dans ces chaînes sont des applications d'édition, de gestion et de restitution du document. Actuellement, l'utilisation du document s'est élargie. Il est notamment utilisé pour communiquer entre applications et systèmes d'information hétérogènes [SOAP 01]. Dans ce mémoire, nous ne faisons aucun *a priori* sur le contenu et sur l'usage du document.

Dans la suite du mémoire, après avoir exposé les problématiques du génie documentaire (cf. section ci-dessous), nous verrons plus en détail les notions introduites dans cette section. La section 2.1 définit un type particulier de représentation de document que sont les documents structurés. L'utilisation de modèles est développée dans la section 2.2, notamment à travers plusieurs exemples. Une étude sur les méta-modèles existants est effectuée dans la section 2.3.

1.2. Problématiques

De nombreux travaux ont été menés dans le domaine du génie documentaire. Pour illustrer la diversité et l'étendue des problèmes actuellement traités, nous donnons ci-dessous plusieurs exemples représentatifs de services de traitement des documents :

- la *dématérialisation des documents* consiste à numériser un document physique (souvent représenté sur un support papier) puis à en extraire une représentation électronique. Une première étape est de reconnaître le document en utilisant essentiellement des techniques de reconnaissance de caractères. Une seconde étape est de structurer l'information reconnue. Le contenu du document est analysé afin d'en déduire l'organisation et les relations entre les composants élémentaires [Watanabe 99] ;

- le besoin de conservation et l'explosion du volume des documents électroniques nécessitent de mettre en place des mécanismes *d'archivage et de recherche*. Alors que ces problématiques ont été largement étudiées dans le domaine des bases de données classiques, de nombreux problèmes et de multiples possibilités apparaissent lorsqu'il s'agit d'archiver et de rechercher des documents [McHugh 97, Salminen 01]. En particulier, les multiples formats de document, souvent propriétaires, posent le problème de l'interprétation des documents encodés dans un de ces formats. Un des

premiers résultats majeurs en matière d'archivage a été la définition de formats standard. Ces standards garantissent la pérennité du format du document ;

- quelque fois, la manipulation d'un document nécessite la présence de plusieurs personnes pour réaliser une tâche commune [Ellis 91]. En particulier, une de ces tâches est *l'édition coopérative* d'un document. Plusieurs personnes, accèdent et modifient un même document de façon distante (les personnes peuvent ne pas être physiquement ensemble) et à des moments différents. Il se pose alors plusieurs problèmes comme la gestion de fragments de document et la perception de la conscience de groupe [Gutwin 99]. Un des résultats sur l'édition coopérative à travers Internet a été normalisé dans Webdav (Web-based Distributed Authoring and Versioning [Webdav 99]). Ce standard décrit des mécanismes de base pour l'accès cohérent à des données partagées à travers le Web ;

- le besoin de *transformation* de document notamment pour les applications d'échange, d'archivage et de présentation de document. Cette technique est au cœur de ce mémoire, c'est pourquoi nous la détaillons dans la section 3.2.1.

En amont de ces fonctions, les travaux de base du génie documentaire sont la modélisation la présentation et l'édition :

- la *modélisation* consiste en la définition de modèles de documents appropriés à un métier donné. Ces métiers peuvent appartenir à des domaines très variés : commerce, droit, chimie, musique, informatique, etc. Le premier niveau de difficulté du problème de la modélisation réside dans la définition d'un modèle qui consiste à définir un vocabulaire couvrant le domaine à modéliser, puis à structurer ce vocabulaire. Ce travail ne peut s'accomplir sans le rapprochement de plusieurs compétences portant sur le domaine concerné et sur la modélisation documentaire, ce qui accroît la difficulté de la tâche de modélisation. La deuxième difficulté se situe au niveau de la définition ou le choix d'un méta-modèle qui permet de :

    - décrire de façon homogène l'ensemble de ces métiers afin de pouvoir réutiliser les traitements associés à un méta-modèle donné ;

    - et de couvrir l'ensemble des besoins de spécification ;

- la *présentation* consiste à rendre perceptible un document à un lecteur et/ou un auteur. Lorsque la représentation du document contient directement des informations de présentation, le processus de présentation consiste à interpréter (ou exécuter) ces informations (cf. figure 1). Dans le cas inverse, le processus de présentation consiste à ajouter des informations de style (graphique, sonore, etc.) et/ou à transformer le document vers un format de présentation. Dans la suite de ce mémoire, nous analysons de façon détaillée ces deux catégories de documents. La première catégorie regroupe les *documents de présentation*, tandis que la seconde catégorie désigne les *documents métier* [Summer 95, Villard 00a]. Les documents qui contiennent à la fois des informations de présentation et des informations métier sont considérés comme étant des documents métier. De façon générale, ils nécessitent en effet une phase d'ajout de style ou de transformation pour être présentés ;

- *l'édition* est la phase durant laquelle la structure et le contenu du document sont créés et mis à jour par un *auteur*. Afin d'augmenter la productivité et l'efficacité de rédaction d'un document, il est indispensable d'assister l'auteur avec plusieurs outils d'aide. Ces outils d'aide à la rédaction peuvent être : le correcteur orthographique, le vérificateur de conformité d'un document vis à vis d'un modèle, l'édition WYSIWYG[3] d'objets textuels, d'objets graphiques, des relations spatiales et temporelles entre ces objets, etc. Une des difficultés de l'édition est la faculté de percevoir les multiples informations contenues dans le document. Ceci est d'autant plus vrai lorsque le document édité est de nature multimédia [Bulterman 95, Jourdan 98].

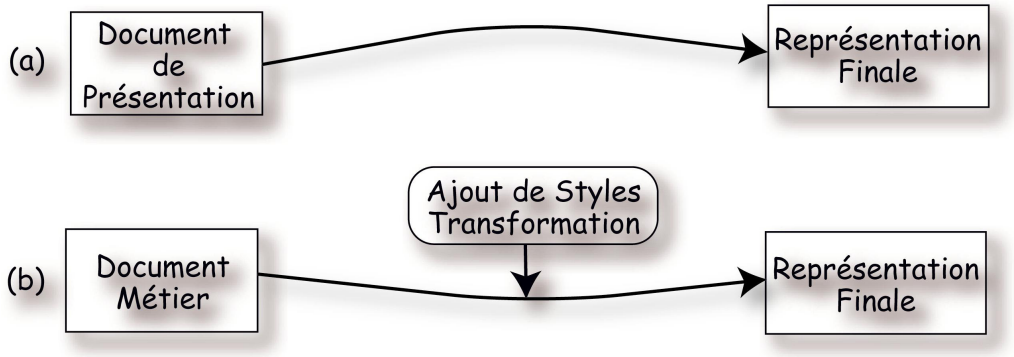


Figure 1. Processus de présentation d'un document de présentation (a) et d'un document métier (b).

Pour traiter des besoins relatifs aux classes de documents multimédias et de l'adaptation au contexte de l'utilisateur, il est indispensable d'étudier plus en détail les techniques associées à la modélisation, à la présentation et à l'édition. En effet, la modélisation de document est à la base de tous les problèmes liés au génie documentaire. Nous étudions ce problème dans la section suivante. Ensuite, puisque le résultat final de ces deux besoins est l'obtention d'une présentation, nous étudions les techniques de présentation existantes. Enfin, jusqu'à maintenant, les systèmes d'édition existants permettent d'éditer des documents multimédias spécifiques et dont la représentation finale est perceptible directement par l'auteur. Cependant qu'en est-il de l'édition de documents lorsque plusieurs présentations finales sont possibles ? Une étude des systèmes d'édition existants, et des techniques sous-jacentes, est donc effectuée dans la section 4 de ce chapitre. Dans la dernière section, nous définissons les objectifs que nous nous sommes fixés dans le cadre des problèmes étudiés dans ce chapitre.

2. Modélisation de documents structurés

Le problème de la modélisation se décompose en trois couches selon le niveau de représentation du document : le document structuré, le modèle et le méta-modèle. Ces couches vont d'une représentation spécifique du document (le document structuré) vers une

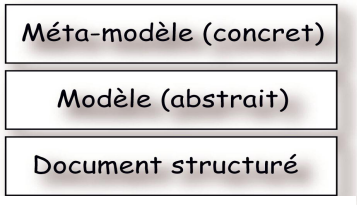


Figure 2. Les trois couches de représentation d'un document.

[3] What You See Is What You Get

représentation générique (le modèle et le méta-modèle). Le méta-modèle permet de représenter formellement un modèle.

2.1. Document structuré

2.1.1. Définition

La notion de document structuré est bien connue de nos jours et les premières définitions datent du début des années 80 [Goldfarb 81]. Nous rappelons simplement le vocabulaire utilisé pour désigner les composants d'un document structuré à travers l'exemple d'un document multimédia illustré figure 3 [Guyard 00]. Ce document est composé d'un *élément* racine de type filmothèque. Cet élément est relié à un ensemble d'éléments film par une relation hiérarchique (père-fils). Chaque film est constitué d'un nom, d'un synopsis, éventuellement d'acteurs et d'une affiche. Ces éléments film et leurs descendants correspondent au *contenu* de l'élément filmothèque.

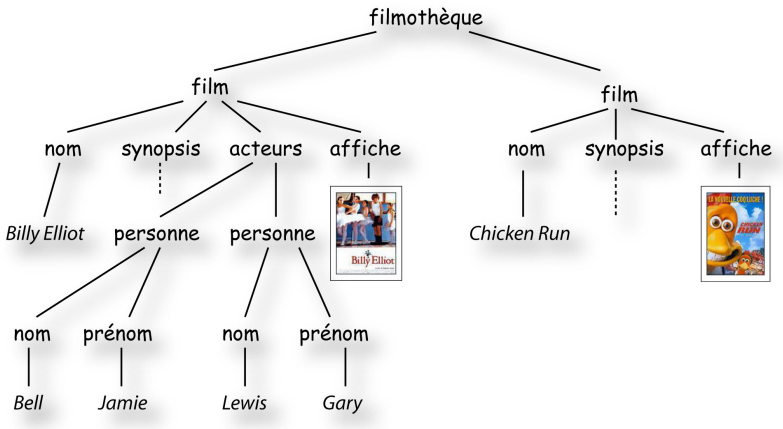


Figure 3. Exemple de document structuré : une filmothèque.

De plus, chaque élément peut éventuellement être caractérisé par un ensemble d'*attributs*. Par exemple, l'élément film est caractérisé par sa catégorie (film d'action, d'aventure, etc.). L'attribut catégorie est alors attaché à ce type d'élément. La valeur est, dans ce cas, de type énuméré : elle correspond à la liste des catégories de film. L'ensemble de ces éléments et de ces attributs constitue la *structure logique* du document structuré représentant une filmothèque.

On peut remarquer que les éléments sont porteurs de sens non seulement par leur type mais aussi par leur contenu et leur position dans la hiérarchie. Par exemple, l'élément nom présent sous l'élément personne est différent de l'élément nom présent sous l'élément film.

2.1.2. Format

Les formats de document structurés sont définis pour stocker la structure logique et le contenu des documents. Le pionnier en matière de format standard est certainement SGML[4] [ISO 86]. Plus récemment, le standard XML[5] [XML 01], défini par le W3C[6], est apparu avec

[4] Standard Generalized Markup Language

[5] eXtensible Markup Language

comme objectif de simplifier et de compléter SGML. Pour ce dernier, il est en effet difficile d'implanter des analyseurs syntaxiques de documents légers et performants pour ce format [Michard 00].

Le principe de ces deux formats est le même : les éléments de structure sont délimités par une *balise* de début d'élément et une balise de fin d'élément. Chaque balise de début d'élément contient le type de l'élément et éventuellement ses attributs. Le contenu de l'élément est situé entre les deux balises. La figure 4 représente dans le format XML le document filmothèque décrit dans la section précédente.

```
1.  <?xml version="1.0" encoding="ISO-8859-1"?>
2.  <filmothèque>
3.      <film catégorie="émotion">
4.          <nom>Billy Elliot</nom>
5.          <synopsis>Billy, onze ans, découvre … </synopsis>
6.          <acteurs>
7.                  <personne>
8.                  <nom>Bell</nom>
9.                  <prénom>Jami</prénom>
10.                 </personne>
11.                 <personne>
12.                     <nom>Lawi</nom>
13.                     <prénom>Gary</prénom>
14.                 </personne>
15.             </acteurs>
16.             <affiche src="billyelliot_poster2.jpg"/>
17.         </film>
18.         <film catégorie="animation">
19.             <nom>Chicken Run</nom>
20.             <synopsis>A la ferme avicole Tweedy, … </synopsis>
21.             <affiche src="chicken_poster2.jpg"/>
22.         </film>
23.     </filmothèque>
```

Figure 4. Représentation d'un document de type « filmothèque » dans le format XML.

Un document XML commence par un prologue composé notamment de la déclaration *<?xml?>*. Celle-ci doit contenir le numéro de version du format XML utilisé dans le document. L'attribut *encoding* permet de préciser le codage de caractères utilisé pour ce document. XML utilise les jeux de caractères définis par la norme ISO 10646 [ISO 00]. Dans ce cas précis, le codage utilisé est l'ISO-8859-1 (iso-latin) : il permet d'écrire les caractères français, en particulier les caractères accentués. Ensuite, le document contient l'arbre d'éléments. Le premier élément correspond à la racine de l'arbre. Dans notre exemple c'est l'élément filmothèque. Les noms d'éléments sont encadrés par des chevrons (<>). La balise ouvrante peut contenir des attributs spécifiés par un nom suivi de sa valeur spécifiée entre guillemets.

[6] World Wide Web Consortium, http://www.w3.org.

Contrairement aux documents SGML, dont la structure est toujours liée à un modèle, le format XML définit deux niveaux de conformité :

- un document est *bien formé* lorsqu'il obéit aux règles syntaxiques du langage XML. Un document bien formé ne fait pas nécessairement référence à un modèle ;
- un document *valide* est bien formé et obéit à un modèle défini dans un méta-modèle.

Le choix de valider ou non un document XML dépend du type d'application utilisant ce document. Ce choix dépend également de la disponibilité ou non du modèle. En effet, de nombreux documents XML existants n'appartiennent pas à un modèle prédéfini.

L'avènement de XML s'est accompagné d'une multitude d'analyseurs syntaxiques. Afin que les analyseurs soient vus comme des composants réutilisables par les applications, ils reposent sur deux standards qui définissent le mode de production du résultat de l'analyse. Ces standards sont indépendants des analyseurs, ils peuvent et sont utilisés par d'autres applications. Le standard DOM [DOM 00] permet aux analyseurs syntaxiques de produire une représentation interne du document XML. Le second standard SAX [SAX] permet aux analyseurs de produire des événements. Ces événements sont envoyés, en particulier, lorsque le début d'un élément ou le contenu d'un élément est analysé. Chaque application, selon ses besoins, peut utiliser l'un ou l'autre mode de production. Par exemple, un éditeur maintient généralement une représentation interne du document XML et donc utilise le premier mode de production. Par contre, un outil de recherche d'un élément particulier dans un document XML utilisera plus probablement le second mode de production.

Le document structuré correspond à un document unique auquel peuvent être associés des traitements spécifiques. Pour pouvoir réutiliser ces traitements sur une collection de documents, la notion de modèle a été introduite.

2.2. Modèles de document

2.2.1. Introduction

Les modèles de documents structurés reposent sur une représentation abstraite qui reflète la structure logique du document. Un document structuré qui correspond à cette structure logique est appelé *instance*. De façon générale, la description de la structure logique d'un document est organisée autour de quatre dimensions [André 89, Layaïda 97] :

- la dimension *logique* permet de lier sémantiquement un ensemble d'éléments. Par exemple, un auteur est caractérisé par un nom et un prénom. Ces liens sont exclusivement hiérarchiques ;
- la dimension *spatiale* concerne l'affectation des objets dans l'espace. Cette affectation concerne les objets visibles en réservant une zone géométrique. Elle concerne aussi les objets audibles pour lesquels un canal audio est alloué ;
- la dimension *temporelle* concerne l'organisation des objets du document dans le temps. Chaque objet est caractérisé par un intervalle de temps pendant lequel il est affiché ou entendu ;

- et la dimension *hypermédia* permet de créer des relations sémantiques, non hiérarchiques, entre des parties de document, comme les renvois et les références. Elle permet aussi de créer des relations de présentation comme des liens de navigation spatiaux ou temporels.

Selon notre classification des documents, un document métier s'exprime uniquement à travers la dimension logique. Par contre, un document de présentation s'exprime à travers une ou plusieurs des dimensions. Un document statique est caractérisé uniquement par les deux premières dimensions. Un document hypermédia est un document statique auquel est ajouté la dimension hypermédia. Lorsqu'un document est caractérisé par l'ensemble des dimensions alors il est considéré comme un document multimédia.

De façon générale, les modèles sont indépendants des traitements appliqués aux documents, et en particulier les modèles métiers sont indépendants de leurs présentations. En effet, que ce soit dans le domaine du génie logiciel, du génie documentaire ou autre, l'automatisation et la réutilisation des traitements sont fondamentales. L'intérêt de spécifier des modèles de document est de pouvoir définir des traitements au niveau générique (modèle) et de pouvoir les appliquer sur chacune des instances, ceci de façon automatique. Par exemple, si une présentation est définie pour un modèle de document qui décrit les formules mathématiques, chaque formule exprimée dans ce modèle peut être présentée en réutilisant cette même présentation.

Ce principe fondamental d'indépendance est cependant à nuancer puisque, souvent, la définition de certains modèles est effectuée pour une application déterminée. Les besoins sont différents selon que le modèle de document est utilisé, par exemple, dans une application d'édition ou une application de recherche d'information. Dans le premier cas, le modèle est conçu de telle façon qu'aucune information ne soit redondante, ceci afin d'éviter d'introduire des incohérences au sein même du document. Par contre, pour les applications de recherche d'information, cette redondance est souvent souhaitée afin d'optimiser la recherche en évitant, par exemple, de traverser des liens hypertextes. Ces modèles sont très proches et le passage de l'un à l'autre peut souvent s'effectuer de façon automatique.

Comme nous l'avons dit précédemment, le document est utilisé dans des domaines très variés : représentation d'articles, commerce électronique, description d'interface utilisateur, représentation de molécules, description de tâche de maintenance aéronautique, etc. Avec l'avènement de XML, de nombreux modèles standard ou non ont été définis, que ce soit pour décrire les documents d'un métier particulier, mais aussi pour décrire des informations de présentation. Nous effectuons un tour d'horizon de ces standards en commençant par les modèles relatifs à un métier. Nous décrivons ensuite les modèles de document représentant des informations de présentation, en les positionnant par rapport aux dimensions énoncées ci-dessus. Le premier objectif de ce tour d'horizon est de familiariser le lecteur avec les modèles utilisés tout au long de ce rapport. Le second objectif est de montrer dans quelle mesure il est possible de générer une présentation adaptée au contexte du lecteur à partir de ces modèles. Le dernier objectif est de mesurer le degré de représentation de l'information multimédia par ces modèles. Nous ne décrivons pas les modèles et les processus qui sont dédiés à l'adaptation dans cette section. Ces modèles et processus seront décrits ultérieurement dans le chapitre 3.

2.2.2. Exemples de modèles métier

Deux modèles de document sont décrits dans les sous-sections suivantes : le modèle Docbook et le modèle ATA 2100. Nous donnons ensuite un bilan sur les caractéristiques communes de ces deux modèles.

2.2.2.1 Docbook

Le modèle de document Docbook [Oasis 01] permet de représenter des livres et des articles en utilisant le format SGML ou le format XML. Ce modèle de document est dans la lignée des travaux effectués par la TEI [TEI] pour représenter l'ensemble des caractéristiques d'un texte. Cependant, Docbook se limite à la description de livres et d'articles, bien qu'il soit utilisé dans un cadre plus large. En particulier, Docbook est utilisé pour la documentation de Linux[7].

Le document figure 4 représente un article décrit en Docbook. L'élément racine article contient un élément artheader et une suite de section. L'élément artheader contient des informations générales sur le document. Par exemple, il permet de lister les auteurs de l'article, de dater le document ou encore de citer les mots clés du document. L'élément section contient un titre et divers éléments. Ces éléments permettent de représenter des paragraphes, des tables ou encore des médias stockés généralement dans des fichiers externes. Par exemple, l'élément videodata (ligne 18) référence une vidéo contenue dans un fichier.

```
1.  <?xml version="1.0" encoding="ISO-8859-1"?>
2.  <article>
3.      <title>Incremental transformation with XSLT</title>
4.      <artheader>
5.          <authorgroup>
6.              <author>
7.                  <firstname>Lionel</firstname>
8.                  <lastname>Villard</lastname>
9.              </author>
10.             </authorgroup>
11.             <date>13 november 2000</date>
12.         </artheader>
13.         <section>
14.             <title>Introduction</title>
15.             <para> …
16.                 <mediaobject>
17.                     <videoobject>
18.                         <videodata fileref="movie.mpg" align="center"/>
19.                     </videoobject>
20.                 </mediaobject>
21.             …
22.             </para>
23.         </section>
24.         …
25.     </article>
```

Figure 5. Exemple d'un article décrit en Docbook.

[7] Linux Document Project, http://www.linuxdoc.org/.

La plupart des éléments et attributs de Docbook sont de nature logique (les éléments section, para, title, etc.). Cependant des informations de présentation peuvent être décrites. La dimension spatiale est implicitement donnée par la structure du document, l'ordre de définition des sections et des paragraphes définit l'ordre conventionnel d'affichage. Quelques attributs, comme l'attribut align, permettent de contrôler la position des éléments. L'élément inlinemediaobject, en particulier, précise que l'objet doit être présenté sur la même ligne que l'élément précédent. À l'opposé, un objet contenu dans l'élément mediaobject doit être affiché sur une nouvelle ligne. La dimension hypermédia est représentée, entre autres, par les éléments xref et ulink. Le premier permet de définir des références croisées à l'intérieur du document et le second permet d'ajouter des liens vers un fichier localisé par une URL[8]. Docbook permet d'inclure plusieurs types de médias (image, vidéo, audio et texte) mais ne permet pas de les synchroniser.

Docbook a été conçu dans un esprit très large et peut être utilisé par de nombreuses applications. Par exemple, la spécification de sections peut se faire soit en utilisant l'élément section récursivement ou les éléments numérotés (sect1, sect2, etc.) définissant leur niveau de profondeur. Le premier élément est adapté à un usage éditorial puisque le déplacement de section ne remet pas en cause le nom de l'élément et de ces descendants. Par contre, les autres éléments peuvent être traités plus rapidement et aussi palier au manque d'expressivité de certains processus incapables de calculer la profondeur de la section (comme le langage de styles CSS1 [CSS1 96]).

2.2.2.2 ATA 2100

La documentation technique aéronautique est constituée de plusieurs manuels, qui sont des documents contractuels qu'un avionneur doit livrer à son client. Chaque manuel respecte un modèle de documents défini par les organismes normatifs comme l'ATA[9] pour l'aviation civile. En particulier, cet organisme définit le modèle de document concernant le manuel de maintenance. Ce modèle est l'ATA 2100 [ATA2100 00]. Le document ci-dessous est un fragment du manuel de l'A320.

```
1. <task chapnbr="26" sectnbr="23" subjnbr="41">
2.     <title>Removal of the Cargo Fire-Extinguisher Bottle</title>
3.     <warning>
4.         <para>DO NOT REMOVE/INSTALL… </para>
5.     </warning>
6.     <topic>
7.         <title>Job Set-up</title>
8.         <subtask func="010" seq="052" key="en26234101005200001">
9.             <list1>
10.                 <litem1>
11.                     <warning>
12.                         <para>BE VERY CAREFUL WHEN YOU MOVE …</para>
13.                     </warning>
14.                     <para>Removal of the Fire Extinguisher Bottle</para>
15.                     <list>
16.                         <litem>
```

[8] Uniform Resource Locator, http://www.w3.org/Addressing/URL/url-spec.html.

[9] Air Transport Association, http://www.air-transport.org/.

```
17.                          <para> Remove the fire extinguisher bottle
18.                             <refint refloc="en26234100000400">26-23-41-000-004
19.                             </refint>.
20.                          </para>
21.                       </litem>
22.                    </list>
23.                 </litem1>
24.              </list1>
25.           </subtask>
26.        </topic>
27.        …
28.    </task>
```

Figure 6. Description d'une tâche de maintenance pour l'A320.

Dans cet exemple, la tâche de maintenance pour enlever un extincteur d'un cargo est décrite. Cette tâche est répertoriée dans le sujet 41 de la section 23 du chapitre 26 du manuel de maintenance. Les messages d'avertissement et de précaution à prendre avant le démontage sont décrits au début de chaque tâche. La description de la tâche même est découpée en plusieurs thèmes (topic). Chaque thème contient une ou plusieurs sous-tâches dont le contenu est structuré, en particulier, sous la forme de paragraphes, de tables et de listes. Les types de médias supportés par l'ATA sont le texte et l'image. L'ATA est donc un format de document métier faiblement multimédia dans le sens qu'il ne supporte que peu de type de médias différents. L'ATA permet de définir des références croisées à l'intérieur d'un même document (l'élément refint) et aussi à l'extérieur (l'élément refext).

De la même manière que pour le modèle Docbook, le modèle ATA est défini pour plusieurs types d'application, en particulier pour la production de manuel et la recherche d'information. Cela se traduit par l'ajout volontaire d'éléments avec la même sémantique mais avec des propriétés structurelles différentes. C'est le cas par exemple des éléments list1, list2, etc. De plus, le contenu des manuels est fortement redondant afin d'être efficace lors de la recherche d'information.

2.2.2.3 Bilan

Les deux modèles présentés ci-dessus sont issus de modèles définis en SGML et adaptés en XML. Cette adaptation s'est faite sans prendre en compte l'ensemble des possibilités de XML notamment sa capacité de définition de modèle en plusieurs modules. Pourtant ces deux modèles utilisent les mêmes concepts, comme les tables, les listes et les paragraphes. Ces concepts sont représentés la plupart du temps par les mêmes éléments (et attributs). Ce n'est pas toujours le cas, comme pour les éléments xref (en Docbook) et refint (en ATA) qui définissent des liens internes.

Les deux modèles sont relativement éloignés de la présentation finale du document. Cependant quelques éléments (et attributs) sont définis pour orienter la présentation du document (align, scale, etc.). De plus, ils sont tous les deux accompagnés d'une spécification informelle sur la façon de présenter ces documents. Lorsque l'auteur a conscience de ces spécifications informelles, il peut s'en servir de plusieurs façons. Par exemple, il peut référencer une image de façon relative en utilisant le mot *ci-dessus* simplement parce que cette

image est déclarée avant. Dans le cadre de l'adaptation de document, ces spécifications sont gênantes puisqu'elles restreignent les possibilités de réorganisation du contenu. Dans l'exemple précédent, l'image ne peut pas être déplacée, ni même supprimée.

2.2.3. Exemples de modèles de présentation

Dans cette section nous décrivons les langages de présentations suivants : CSS, XHTML, SMIL 2.0, SVG et Madeus. Hormis CSS, tous ces langages sont syntaxiquement décrits en XML.

2.2.3.1 CSS

CSS (Cascading Style Sheet [CSS2 01]) est un langage qui permet d'attacher du style à des documents structurés. Une feuille de style CSS est constituée d'un ensemble de règles de style. Par exemple, la règle suivante spécifie une couleur bleue et une taille de caractères de 12 points pour les éléments para qui sont des fils de l'élément h1 :

```
1. h1 > para { color : blue; font-size: 12pt }
```

Ces règles se décomposent en deux parties : le *sélecteur* (h1 > para) et les *déclarations* entre accolades. Un sélecteur permet d'identifier un ensemble d'éléments sur lesquels les déclarations sont appliquées. Une déclaration est composée d'une propriété (color) et d'une valeur fixe (blue). Plusieurs déclarations peuvent être spécifiées pour un même sélecteur en les séparant par le caractère ';'.

Une des caractéristiques importante de CSS est qu'il permet de spécifier des styles pour différents types de support : sur l'écran, sur papier, avec un synthétiseur vocal, avec un lecteur de braille, etc. Les propriétés s'appliquent ou non selon le type de support. Une autre caractéristique introduite dans CSS2 est la génération du contenu par la feuille de style. Par exemple, CSS2 permet de générer des numéros de section. Ce contenu peut être soit inséré avant l'élément sélectionné ou soit après.

La majorité des propriétés CSS ne contrôlent pas l'organisation des objets du document. Néanmoins, lorsque la feuille de style est destinée à un support de type visuel, CSS définit une organisation graphique des objets du document. Dans ce cas, le modèle de placement CSS est un modèle de boîtes organisées les unes en dessous des autres. Ce mode de placement est complété par un positionnement absolu.

CSS est un langage qui est conçu à la base pour être très simple. Il couvre de nombreux besoins pour caractériser les styles avec la définition de plus d'une centaine de propriétés. Et il est très largement utilisé notamment par de nombreux documents qui utilisent les propriétés CSS comme valeur d'attribut. De plus, comme les sélecteurs permettent d'identifier des éléments selon leur type, les feuilles de style peuvent s'appliquer à n'importe quel document XML.

2.2.3.2 XHTML

La spécification XHTML [XHTML 00] est la reformulation en XML du langage HTML [HTML 99]. Cette spécification définit le langage de publication de document sur le Web.

Pour des raisons d'efficacité et historique, ce langage est constitué d'un mélange d'éléments logiques et d'éléments de présentation.

Les éléments logiques permettent de définir des paragraphes, des listes, etc. La dimension hypermédia est présente grâce aux éléments a et area. La présentation spatiale est dirigée par la structure du document et paramétrée par des attributs de style présents sur les éléments. De plus, quelques éléments ont une sémantique de présentation : frame, i, b, etc. Des informations de style CSS peuvent être associées à chaque élément du langage, quelque soit leur dimension. La figure ci-dessous donne un exemple très simple de document XHTML.

```
1.  <?xml version="1.0" encoding="ISO-8859-1"?>
2.  <html xmlns="http://www.w3.org/1999/xhtml">
3.      <head>
4.          <title>Rapport d'avancement</title>
5.      </head>
6.      <body>
7.          <h1 class="title">
8.                  <a name="N2">Rapport d'avancement</a>
9.          </h1>
10.             <h2 class="author">Lionel Villard</h2>
11.             ...
12.         </body>
13.     </html>
```

Figure 7. Exemple de document XHTML.

À chaque élément XHTML est associée de façon informelle une sémantique de présentation. C'est à l'application de respecter ou non cette sémantique pour les éléments logiques. Cependant, en ce qui concerne les éléments de présentation, l'application est contrainte au respect de la sémantique de présentation associée.

2.2.3.3 SMIL 2.0

Le modèle de document SMIL 2.0 [SMIL 01] décrit principalement l'organisation temporelle de médias. Chaque nœud composite, encore appelé opérateur, porte une sémantique temporelle qui définit le placement temporel des éléments contenus dans l'opérateur :

- l'élément seq permet de présenter les médias en séquence ;
- l'élément par permet de présenter les médias en parallèle ;
- l'élément excl permet de présenter les médias de façon exclusive : un seul média est joué parmi l'ensemble des médias définis dans l'élément excl.

Ces opérateurs permettent de synchroniser les éléments de façon relativement grossière. Afin de raffiner cette synchronisation, SMIL permet de spécifier des arcs de synchronisation (des événements). Ces arcs de synchronisation sont spécifiés grâce aux attributs begin et end attachés aux médias et aux opérateurs. Voici un exemple de document SMIL :

```
1.  <?xml version="1.0" encoding="ISO-8859-1"?>
2.  <smil xmlns="http://www.w3.org/2000/SMIL20/CR/Language">
```

```
3.      <head>
4.          <layout>
5.              <region id="R" left="10" .../>
6.          </layout>
7.      </head>
8.      <body>
9.          <seq id="main">
10.             <img id="back" src="background.gif" region="R"/>
11.             <audio id="sound" src="song.wav" begin="id(back)+1s"/>
12.         </seq>
13.     </body>
14. </smil>
```

Figure 8. Exemple de document SMIL 2.0.

La sémantique temporelle de ce document est la suivante : deux objets back et sound se jouent en séquence. L'objet de type audio sound se joue une seconde après la fin de l'objet de type image back grâce à la spécification de l'arc de synchronisation begin="id(back)+1s".

On peut noter que SMIL contient aussi un ensemble d'éléments dédiés à l'organisation spatiale (lignes 4 à 6). Chaque média visuel est associé à une région spatiale (élément region). Le positionnement de la région s'effectue de façon absolue. De plus, SMIL2.0 permet d'imbriquer les régions pour spécifier des régions positionnées de façon relative. Cependant, contrairement au modèle temporel, aucun opérateur spécifiant une organisation des médias dans l'espace n'est défini.

La spécification du langage SMIL2.0 est découpée en plusieurs *modules*. Chaque module regroupe un ou plusieurs éléments et/ou attributs caractérisant un trait particulier du langage. Par exemple, le module appelé « animation basique » regroupe les éléments animate, set, animateMotion et animateColor qui permettent de spécifier des animations de base sur les médias. L'avantage de ce découpage est de pouvoir réutiliser une partie de la spécification pour divers besoins. Nous verrons quelques exemples dans la suite de ce mémoire.

2.2.3.4 SVG

Les documents SVG [SVG 01] représentent des dessins vectoriels. Le principal nœud composite est l'élément **g** qui permet de grouper des objets graphiques. Leur système de coordonnées est alors relatif à leur élément père **g**. L'exemple suivant correspond à un fragment du document SVG dont l'interprétation est illustrée dans la section 1.2 figure 1 :

```
1.  <?xml version="1.0" encoding="iso-8859-1"?>
2.  <svg width="596.711pt" height="195.262pt" viewBox="0 0 596.711 195.262">
3.      <g style="font-family:'ComicSansMS'; font-size:17.6">
4.          <path style="fill:#FFFFFF" d="M292.959 …"/>
5.          <path style="fill:#FFFFFF" d="M463.495,65.09v61 …"/>
6.          <path style="fill:#FFFFFF" d="M34.535,133.05v61 …"/>
7.          <path style="fill:#FFFFFF;" d="M36.044,0.5v61 … "/>
8.          <text>
9.              <tspan x="0" y="0">Document </tspan>
10.                 <tspan x="33.125" y="19.2">de</tspan>
```

```
11.                 <tspan x="-8.723" y="38.4">Présentation</tspan>
12.             </text>
13.         …
14.     </svg>
```

L'élément svg permet de décrire un dessin vectoriel auquel sont associés la dimension du dessin ainsi qu'un système de coordonnés (attribut viewBox). La description même du dessin vectoriel est défini dans le contenu de l'élément svg. Cet exemple comporte deux types d'objets graphiques, l'objet path et l'objet text. Un objet path permet de définir un objet graphique ayant une forme arbitraire dont le contour est spécifié dans l'attribut d. L'objet text permet d'ajouter du texte dans le dessin vectoriel. Cet objet peut contenir éventuellement des éléments tspan permettant de décaler horizontalement et verticalement des fragments de texte. À chacun des éléments, il est possible d'associer du style CSS. Lorsque ce style est spécifié sur un élément composite, la valeur est alors propagée sur les descendants du composite.

En plus de la spécification du placement spatial des objets graphiques, il est possible de changer le graphique vectoriel à travers le temps et donc d'offrir un moyen de décrire des animations en utilisant le module d'animation défini dans SMIL 2.0 (cf. section précédente). L'exemple suivant décrit un effet visuel de type *fade out* (variation de l'attribut *opacity*), c'est-à-dire que l'objet rectangle disparaît progressivement pour devenir totalement invisible au bout de cinq secondes :

```
1.  <rect>
2.      <animate attributeType="text/css" attributeName="opacity"
                    from="1" to="0" dur="5s"/>
3.  </rect>
```

La spécification temporelle est localisée sur un attribut d'un objet graphique. Elle est similaire à l'approche événementielle de SMIL. Il est possible de synchroniser le début de l'animation avec la fin d'une autre animation. Par exemple l'animation suivante commence 45 secondes après la fin de l'animation x :

```
1.  <animate begin="x.end+45s"/>
```

Par contre, l'information temporelle n'est pas structurée. Il n'est pas possible de profiter de la sémantique des opérateurs. Il n'est donc notamment pas possible de spécifier l'exclusion d'animation comme le permet SMIL grâce à l'opérateur excl.

2.2.3.5 Madeus 1.0

Madeus 1.0 est un langage de description de présentations multimédias défini par Nabil Layaïda Le modèle sous-jacent de ce langage est basé sur des intervalles de temps [Layaïda 97]. Le format de Madeus permet de décrire l'organisation logique ainsi que l'organisation temporelle, spatiale et hyperlien des médias :

- l'organisation logique est décrite grâce à l'élément composite qui définit un objet par la composition d'autres objets composites et/ou des objets de base ;

- les autres organisations sont définies par les types d'objets et les attributs attachés aux objets composites ou aux objets de base. Les attributs expriment des propriétés temporelles (durée), graphiques ainsi que les informations de liens (ancre destination). L'élément relation permet de spécifier les relations temporelles et les relations spatiales. Il ne peut être attaché qu'aux objets composites et il n'est donc pas possible de placer des relations au sein même d'un média.

Par exemple, le document suivant montre la spécification d'un scénario Madeus qui contient un composite appelé Histoire. Ce composite est composé de trois objets (audio, image et texte) avec des relations temporelles exprimant le scénario suivant : les trois objets commencent et finissent en même temps. De même, la partie marquée par l'élément spatial définit les relations spatiales entre l'image et le texte. Les deux objets doivent être alignés par le haut et le texte est à une distance de 10 points à droite de l'image.

```
1.  <?xml version="1.0" encoding="iso-8859-1"?>
2.  <madeus>
3.      <composite name="Histoire" fontFamily="times" fontSize="14">
4.          <audio name="Audio" source="histoire.au" duration="30 35 40"/>
5.          <image name="Image" source="affiche.jpg" duration="10 35 100"
                    left="20" top="40"/>
6.          <text name="text" source="histoire.html" fontSize="16"/>
7.          <relations>
8.              <temporal>
9.                  <equals interval1="Audio" Interval2="Image"/>
10.                     <equals interval1="Image" Interval2="Text"/>
11.                 </temporal>
12.                 <spatial>
13.                     <topAlign interval1="Image" interval2="Text"/>
14.                     <rightSpacing interval1="Image" interval2="Text" distance="10"/>
15.                 </spatial>
16.             </relations>
17.     </madeus>
```

Madeus se distingue de SMIL essentiellement par les caractéristiques suivantes :

- le positionnement temporel et spatial peut s'effectuer de façon relative en utilisant des relations (cf. exemple précédent) ;

- 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.

Ces différences font que Madeus est plus complexe à traiter par les applications de type lecteur (cf. section 3.2.2).

2.2.4. Exemple d'un modèle « sémantique » : RDF

RDF est un ensemble de spécifications, initié par le W3C, pour la représentation des méta-données intégrant une sémantique forte. La spécification du modèle et de la syntaxe RDF [RDF 99] fournit un système pour représenter des ontologies servant à augmenter l'interopérabilité et les échanges entre applications.

RDF est un graphe orienté étiqueté dont la syntaxe est écrite en XML. Il s'agit d'un formalisme utilisé pour représenter les propriétés d'une ressource et les valeurs de ces propriétés. Il contient donc trois types d'objets : les *ressources*, les *propriétés* et les *valeurs*. Une ressource peut être une page HTML, une partie d'une page HTML, un ensemble de pages, un objet ou une entité qui peut être accédée par une URI. L'exemple de la figure 9 montre une ressource nommée Hardware qui est caractérisé par deux propriétés Mémoire et Ecran. Ces propriétés ont respectivement comme valeur 32MB et 800*600*24.

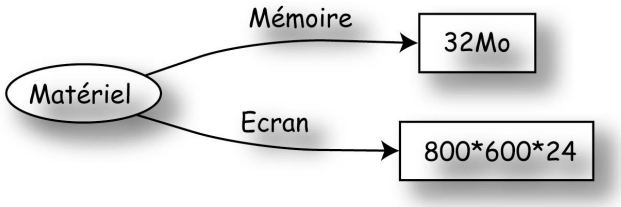


Figure 9. Exemple simple de modèle RDF.

En XML, ce modèle se spécifie comme suit :

```
1. <?xml version="1.0" encoding="ISO-8859-1">
2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.           xmlns:ccpp="http://www.w3.org/2000/07/04-ccpp#">
4.    <rdf:Description about="Matériel">
5.       <ccpp:Mémoire>32MB</ccpp:Mémoire>
6.       <ccpp:Ecran>800*600*24</ccpp:Ecran>
7.    </rdf:Description>
8. </rdf:RDF>
```

En plus du besoin de définir la syntaxe du modèle RDF en utilisant un méta-modèle, le W3C propose un langage pour définir la sémantique des modèles RDF : RDFS [RDFSchema 00]. Ce travail de spécification est en cours au sein du W3C c'est pourquoi nous avons choisi de ne pas le décrire.

2.2.5. Bilan sur les modèles

Dans cette section, nous avons présenté plusieurs types de document de façon indépendante. Ces modèles doivent pouvoir être combinés ensemble pour être traités par différentes applications et/ou pour présenter des documents plus complexes. Nous avons vu un exemple avec SVG et le module d'animation de SMIL 2.0. Un document de type Docbook peut naturellement, du point de vue de l'auteur, contenir des dessins vectoriels décrits en SVG et des formules mathématiques décrites en MathML. Dans le monde XML, cela est possible grâce aux espaces de noms [Namespace 99]. Cependant, afin d'obtenir une présentation cohérente, cette intégration doit être contrôlée. Cela n'a pas de sens, par exemple, d'insérer une formule mathématique à l'intérieur de l'élément `author` de Docbook. Cette intégration est définie de façon similaire à la définition d'un modèle : en utilisant un méta-modèle.

Les modèles de présentation décrits dans cette section, sauf Madeus, permettent de spécifier des présentations relativement rigides. En particulier, concernant la dimension spatiale, le positionnement et la taille des objets sont fixés par une valeur unique. Seuls quelques langages permettant de représenter des objets textuels offrent une certaine souplesse de présentation.

Cette souplesse provient du fait que le texte peut être présenté sur plusieurs lignes. Concernant les langages que nous avons décrits, seul SVG ne permet pas une telle souplesse. Un des avantages d'avoir des présentations rigides est de simplifier la tâche de l'interpréteur de la présentation.

Les modèles de document sont de plus en plus complexes. C'est pourquoi ces modèles ont tendance à être définis de façon modulaire à la fois pour permettre de réutiliser les concepts et aussi pour répondre en partie au problème d'adaptation. Les périphériques d'accès à l'information sont souvent limités en ressource CPU et mémoire et c'est pourquoi des modèles plus légers ont été définis à partir d'un sous-ensemble de modules composant un langage. Par exemple, le profil SMIL Basic a été défini à partir du langage SMIL 2.0 pour jouer de « petits » documents multimédias. Dans le même esprit, le profil XHTML basic a été défini à partir des modules XHTML 1.0. Bien que ces profils allègent le traitement pour les présenter, ils ne prennent pas en compte les autres caractéristiques des périphériques de type PDA, en particulier leur faible espace visuel. De plus, ils nécessitent un travail plus important lors de la conception du document.

Les modèles correspondent à une notion abstraite représentant une classe de document. Pour identifier qu'un document appartient à une classe, c'est-à-dire qu'il correspond à un modèle, il est nécessaire d'exprimer formellement le modèle. C'est l'objectif des méta-modèles que nous décrivons dans la section suivante.

2.3. Méta-modèle

Dans cette section, nous commençons par donner les principes généraux relatifs aux méta-modèles et nous citons les langages existants pour créer de tels méta-modèles. Nous proposons ensuite plusieurs critères afin de les comparer et aussi pour en faire ressortir les principales caractéristiques. Pour terminer, dans la dernière sous-section nous décrivons plus en détail trois de ces langages.

2.3.1. Principes généraux

Comme nous l'avons vu précédemment, un modèle de document est défini formellement par un méta-modèle. Un méta-modèle permet d'exprimer des contraintes à la fois sur la structure et sur la valeur des attributs des documents appartenant à ce modèle. Par exemple, une contrainte imposée par le modèle filmothèque est qu'un élément `film` doit contenir un élément `nom` représentant le nom du film.

L'application principale des méta-modèles est la *validation*, c'est-à-dire la vérification qu'un document structuré correspond à un modèle donné. Le résultat de la validation est binaire, soit le document est valide, soit au contraire il est invalide. Dans ce dernier cas, un message de diagnostic peut être envoyé à l'utilisateur. Quelques-uns des méta-modèles existants permettent de spécifier ce message (cf. section suivante). Une autre application des méta-modèles est l'aide à l'édition. Par exemple, plusieurs outils d'édition textuelle proposent de compléter le nom de l'élément en cours d'écriture en proposant uniquement la liste des éléments valides.

Les premiers travaux sur la conception de méta-modèles datent du milieu des années 80. On peut citer les travaux de Vincent Quint sur la définition du langage S [Quint 87], ainsi que la définition du standard SGML [ISO 86] dont on connaît le succès aujourd'hui à travers les

standards du Web. C'est avec l'arrivée de XML qu'une forte activité est ressentie autour de la définition de méta-langages expressifs et performants. Cela s'est traduit en peu de temps par l'apparition d'une dizaine de nouveaux langages. Nous donnons une brève description de ces langages ci-dessous :

- **DTD XML :** les DTD XML est le standard de fait puisqu'il est défini dans la norme XML [XML 01]. C'est un langage simple qui a de nombreuses limites par rapport aux autres méta-modèles. C'est aussi le seul langage qui ne repose pas sur la syntaxe XML ;

- **DSD** : DSD (Document Structure Description) est co-développé par le laboratoire AT&T et BRICS. Leur objectif est de permettre une description du contenu et des attributs qui dépendent du contexte, d'être proche de l'expressivité de XSLT, de permettre la redéfinition de classes existantes, etc. [Klarlund 00] ;

- **Osmos** : Osmos est un projet de l'EPFL[10] qui contient un langage de modélisation. Ce langage est focalisé sur la réutilisation de modèle ou de fragment de modèle [Rekik 01] ;

- **Relax NG** : Relax NG est le résultat de la fusion de Relax [Murata 01] et de TREX [Clark 01a]. Un schéma Relax spécifie un patron pour la structure et le contenu des documents XML. L'objectif de Relax NG est d'être simple, facile à apprendre, de traiter les attributs et les éléments de façon uniforme, etc. [Clark 01b] ;

- **XML Schema** : XML Schema [Schema 01] est la réponse du W3C au manque d'expressivité des DTD XML. Ce langage est détaillé dans la section 2.3.3.2 ;

- **Schematon** : Schematron [Jelliffe01], développé par Rick Jelliffe, aborde la conception d'un méta-modèle de façon totalement différente par rapport aux autres méta-modèles. En effet, Schematron se focalise sur la validation d'un modèle plutôt que sur la définition d'un modèle. Cette singularité nous a conduit à le décrire de façon plus précise dans la section 2.3.3.3 ;

- **XDR** [Microsoft 01a] : XDR (XML-Data Reduced) est le méta-langage de Microsoft utilisé dans BizTalk. Ce langage est un sous-ensemble de XML Schema hormis le fait que XDR est un modèle ouvert (cf. section 2.3.2.4). En outre, il existe une feuille de transformation XSLT pour convertir des schémas XDR en XML Schema [Microsoft 01b] ;

- **SOX :** SOX (Schema for Object-Oriented XML) est développé par Commerce One. La conception de ce langage a été très fortement influencée par le paradigme orienté-objet et inclut le concept d'interface et d'implémentation [SOX 99].

Afin de comparer ces langages, nous avons sélectionné un ensemble de critères caractérisant ces méta-langages dont certains sont issus de la liste de diffusion XML [XML-list].

[10] Ecole Polytechnique Fédérale de Lausanne, http://www.epfl.ch/

2.3.2. Critères de comparaison

2.3.2.1 Expressivité

Les méta-modèles permettent de spécifier certains types de contraintes. Par exemple, seuls Schematron et DSD permettent d'exprimer des conditions pour imposer, par exemple, qu'un attribut doit être spécifié si la valeur d'un autre attribut est égale à une valeur précise. Une liste complète des types de contrainte pouvant être exprimés par les méta-modèles existants a été établie par Dongwon Lee et Wesley W. Chu [Lee 00]. Nous en donnons un extrait ci-dessous :

- au niveau des types de données. Deux catégories de types de données sont identifiés : simple ou complexe. Un type simple ne peut pas définir un contenu en fonction d'autres éléments et d'autres attributs. La définition de types complexes, au contraire, permet ce genre de définition ; les contraintes possibles à ce niveau sont étudiées dans les deux derniers points. Les contraintes répertoriées sur les types simples sont les suivantes :
    - types prédéfinis : ce sont les types primitifs ou dérivés simples fournis par le méta-modèle. Par exemple XML-Schema propose 37 types prédéfinis, tandis que DSD n'en propose aucun ;
    - définition de type par l'utilisateur : le méta-modèle peut fournir un moyen pour définir de nouveaux types simples à partir des types de base. Seuls XML-Schema et DSD proposent cette fonctionnalité ;
    - limitation du domaine de valeur : quelques méta-modèles proposent un ensemble de constructeurs pour limiter le domaine de valeurs valides pour les types de données. Tous les méta-modèles permettent de spécifier ce type de contrainte, hormis les DTD XML ;
- au niveau des attributs :
    - choix parmi plusieurs attributs : cette fonctionnalité permet d'associer plusieurs attributs à un élément et de contraindre le choix à un seul attribut au niveau de l'instance ;
    - définition conditionnelle : un attribut a_1 d'un élément e est pertinent seulement lorsqu'un attribut a_2 contient une certaine valeur. Seuls Schématron et DSD permettent de spécifier ce type de contrainte ;
- au niveau des éléments :
    - liste ordonnée / non ordonnée : tous les langages permettent de spécifier des listes ordonnées d'éléments. Par contre seuls quelques-uns offrent la possibilité de spécifier des listes non ordonnées, comme XML Schema et Schématron ;
    - spécification minimum et maximum de l'occurrence : quelques méta-modèles permettent de spécifier le nombre minimum et maximum d'éléments qui peuvent apparaître dans le contenu d'un élément ;

- héritage :
    - type simple par extension : de nouveaux types simples peuvent être créés à partir des autres types simples en relâchant les contraintes sur le domaine de valeur. Aucun méta-langage existant n'offre cette possibilité ;
    - type simple par restriction : le domaine de valeur du nouveau type est un sous-ensemble du domaine du type de base ;
    - type complexe par extension : permet d'étendre un type complexe. Généralement, la dérivation s'effectue en ajoutant des éléments à la fin de l'élément de base ;
    - type complexe par restriction : permet de restreindre un type complexe ;
- divers :
    - contrainte dynamique : Schematron permet d'activer ou de désactiver les contraintes à utiliser durant la phase de validation ;
    - documentation : plusieurs niveaux de documentation peuvent être offert. Tous les méta-langages permettent de documenter le modèle au sein même de la définition. XDR, les DTD et Osmos ne permettent pas de spécifier des messages de diagnostic lorsque la validation échoue.

2.3.2.2 Modularité

Ce critère mesure la capacité d'un langage à pouvoir être défini de façon modulaire. Un langage modulaire facilite la réutilisation d'une partie d'un modèle ainsi que sa maintenance. De plus, une modélisation modulaire est généralement plus lisible. Nous avons vu une application concrète d'utilisation de modules à travers la définition de profils langagiers (cf. section 2.2).

Deux types d'inclusion de fragments de schémas peuvent être considérées. Le premier type consiste en l'inclusion d'un fragment dans le même espace de nom que le schéma courant. SOX, XML Schema et DSD supportent ce type d'inclusion. Le second type consiste au contraire d'inclure un fragment dans un espace de nom différent. XML Schema et SOX offrent cette fonctionnalité.

2.3.2.3 Grammaire versus règle

La plupart des méta-modèles repose sur le concept de grammaire régulière pour spécifier les contraintes. C'est le cas en particulier des DTD XML, des schémas XML, de Relax NG et de OSMOS. Une deuxième catégorie de méta-modèles concerne ceux basés sur le concept de règle. C'est en particulier le cas de Schematron et en partie de DSD. Une des différences fondamentales entre ces deux catégories est que l'approche grammaire *définit* un modèle, tandis que les règles décrivent seulement comment *valider* un document vis-à-vis d'un modèle. De plus, on peut noter que :

- les langages basés sur une grammaire sont moins expressifs que les langages à base de règles. En effet, certaines contraintes sont difficiles à exprimer dans les grammaires régulières. Par exemple, l'expression de contraintes contextuelles est difficile à exprimer,

en particulier, lorsqu'un élément a un parent P alors cet élément doit avoir un attribut A. Un autre exemple est l'expression de contraintes co-occurrentes : si un élément a un attribut A, alors cet élément doit aussi avoir un attribut B. Seuls Schematron et DSD permettent de spécifier cette dernière contrainte ;

- d'un autre côté, les langages basés sur une grammaire sont d'un niveau sémantique plus élevé que les langages à base de règles. Par exemple, dans la figure 10 la même contrainte de choix entre deux éléments est exprimée en Schematron et en XML-Schema. Schematron nécessite deux assertions. La première spécifie que l'élément personne doit avoir au moins un fils de type homme ou femme, et la seconde contraint le nombre de fils. En conclusion, un langage à base de règles requiert une analyse préalable pour en déduire des contraintes structurelles de haut niveau.

```
1. <rule context="personne">
2.     <assert test="homme or femme"/>
3.     <assert test="count(*) = 1"/>
4. </rule>
```

```
1. <element name="personne">
2.     <choice>
3.         <element ref="homme"/>
4.         <element ref="femme"/>
5.     </choice>
6. </element>
```

Figure 10. Différences entre Schematron et XML-Schema pour exprimer une contrainte de type choix d'éléments.

2.3.2.4 Ouvert versus fermé

Un langage est dit ouvert lorsqu'il permet la présence d'éléments et d'attributs supplémentaires dans un élément sans avoir le besoin que ces types ou attributs aient été déclarés. C'est en particulier le cas de Schematron et XDR. Les langages ayant un comportement contraire sont dits fermés.

2.3.2.5 Performance lors de la validation

Le temps de validation varie selon le méta-modèle employé. De la même façon qu'il existe deux types de résultat issus de l'analyse syntaxique d'un document XML (cf. section 2.1.2), deux approches sont possibles pour valider un document : par arbre et par événement [Murata 01]. Dans la première approche, le validateur s'appuie sur la représentation interne et complète du document à valider. Dans la seconde approche, la validation s'effectue lors de la réception des événements produits lors de l'analyse syntaxique du document XML. Ces deux approches permettent de valider de petits documents. Par contre, plus le document à valider est gros, et moins la seconde approche est adaptée.

2.3.3. Description de méta-modèles existants

Dans cette section, nous présentons quelques-uns de ces langages. Le choix de ces langages repose sur les faits suivants. Les langages basés sur le concept de grammaire se distinguent principalement par leur expressivité. La DTD XML est le langage le moins expressif mais constitue le langage de référence puisqu'il est largement adopté. À l'opposé, XML Schema est probablement le langage le plus expressif [Lee 00]. Schematron est un langage original

puisqu'il est basé sur le concept de règles et que c'est un langage ouvert. Nous décrivons ces trois langages dans les sous-sections suivantes.

2.3.3.1 DTD XML

La recommandation XML définit non seulement le format d'une instance de document structuré mais aussi la syntaxe pour définir des types de document. C'est à l'heure actuelle le méta-modèle le plus utilisé puisqu'il fut longtemps le seul disponible. La déclaration d'un élément s'effectue en utilisant l'instruction `<!ELEMENT nomElement contenu>` avec *nomElement* correspondant au nom de l'élément en cours de définition et *contenu* le contenu de cet élément. Les constructeurs utilisés pour définir le contenu sont les suivants :

- le mot clé EMPTY qui indique que l'élément n'a aucun contenu :
- le mot clé ANY qui indique que l'élément peut contenir n'importe quel contenu ;
- le mot clé #PCDATA qui indique la présence d'une suite de caractères ;
- l'agrégat représenté par le caractère ',' qui définit un ensemble ordonné d'éléments de même type ou de type différent ;
- la liste qui définit un ensemble d'éléments de même type. Le caractère '+' indique que la liste doit contenir au moins un élément, le caractère '*' indique que la liste peut contenir zéro ou plusieurs éléments et le caractère '?' qui indique que la liste contient zéro ou un élément ;
- le choix, représenté par le caractère '|' qui définit une alternative entre plusieurs types d'éléments.

En plus de la déclaration d'éléments, les DTD XML permettent de définir des attributs en utilisant la déclaration `<!ATTLIST nomElement nomAttribut type défaut>`. Le type correspond à un des 10 types de base prédéfinis, comme `CDATA` pour le type chaîne de caractère. `Défaut` permet soit d'indiquer la valeur par défaut de l'attribut, soit de caractériser si l'attribut est obligatoire (`#REQUIRED`) ou facultatif (`#IMPLIED`).

Voici un fragment de la DTD filmothèque exprimé dans la syntaxe décrite ci-dessus :

```
1. <!ELEMENT filmothèque (film*)>
2. <!ELEMENT film (nom, acteurs?, affiche?)>
3. <!ATTLIST film catégorie CDATA #REQUIRED>
```

Comme nous pouvons le constater, les DTD XML ont un pouvoir d'expression très limité. De nombreuses contraintes ne peuvent pas être exprimées. Il n'est ainsi pas possible de spécifier le nombre d'occurrences d'un élément, ni de définir des conditions sur l'existence ou non d'un attribut. Pour palier à ces limites, le W3C propose les schémas XML.

2.3.3.2 Schéma XML

Les schémas XML représentent une évolution majeure pour définir des modèles. C'est le langage qui propose le plus de type de base, avec 37 types. De plus, quasiment la totalité des contraintes décrites dans la section 2.3.2.1 peuvent être exprimées. En contrepartie, cela en fait

un langage complexe, nécessitant plus de 300 pages pour en décrire tous les traits. Dans cette section, nous proposons un aperçu de ce langage à travers l'exemple ci-dessous :

```
1. <?xml version="1.0" encoding="ISO-8859-1"?>
2. <schema xmlns="http://www.w3.org/2001/XMLSchema">
3.
4.     <element name="filmothèque">
5.         <complexType content="elementOnly">
6.             <element ref="film" maxOccurs="unbounded"/>
7.         </complexType>
8.     </element>
9.
10.        <element name="film">
11.            <complexType content="elementOnly">
12.                <attribute name="catégorie" type="catType"/>
13.                <xsd:sequence>
14.                    <element ref="nom" minOccurs="1" maxOccurs="1"/>
15.                    <element ref="acteurs" minOccurs="0" maxOccurs="1"/>
16.                    <element ref="affiche" minOccurs="0" maxOccurs="1"/>
17.                </xsd:sequence>
18.            </complexType>
19.        </element>
20.
21.        <element name="nom" type="xsd:string"/>
22.        …
23.    </schema>
```

Figure 11. Fragment du schéma filmothèque.

Les éléments sont définis grâce à l'élément element contenant un attribut name spécifiant le nom de l'élément. Les éléments simples, comme l'élément déclaré ligne 15 de la figure 11, se définissent en ajoutant l'attribut type dont la valeur correspond à un des 37 types prédéfinis par la norme. Dans ce cas précis, le contenu de l'élément nom est une chaîne de caractères (xsd:string). Les éléments complexes sont spécifiés en utilisant l'élément complexType (lignes 5 et 11). Cet élément permet de définir des attributs (ligne 12) et des éléments ou des références vers des éléments existants (lignes 6, 13, 14, 15). Le nombre d'occurrence des éléments peut être spécifié en utilisant les attributs minOccurs et maxOccurs.

Une des caractéristiques intéressantes est de pouvoir restreindre ou augmenter les types de base et les éléments complexes. Par exemple, le type simple catType (ligne 12) est défini comme suit :

```
1. <xs:simpleType name="catType">
2.     <xs:restriction base="xsd:string">
3.         <xs:enumeration value="émotion"/>
4.         <xs:enumeration value="action"/>
5.         <xs:enumeration value="animation"/>
6.         …
7.     </xs:restriction>
8. </xs:simpleType>
```

Dans cet exemple, le type simple catType est défini en restreignant le type chaîne de caractères à seulement quelques chaînes relatives à la catégorie d'un film.

2.3.3.3 Schematron

Schematron [Jelliffe01] est un langage qui repose sur le concept de règle. Ce langage est ouvert, c'est-à-dire que par défaut il n'impose aucune restriction ni sur les types d'élément, ni sur l'organisation de ces éléments dans un document ; c'est le schéma qui restreint le modèle de document.

Un schéma Schematron respecte un modèle de document. La DTD XML partielle (sans la définition des attributs) de ce modèle est la suivante :

```
1. <!ELEMENT schema (title ?, phase*, pattern+)>
2. <!ELEMENT phase (active*)>
3. <!ELEMENT pattern (p*, rule*)>
4. <!ELEMENT rule ( (assert | report)* )>
5. <!ELEMENT assert  (#PCDATA)>
6. <!ELEMENT report  (#PCDATA)>
7. <!ELEMENT p (#PCDATA)>
```

La sémantique de ces éléments est la suivante :

- **schema** : c'est l'élément racine du langage ;
- **assert** : cet élément est utilisé pour ajouter des assertions sur le document. Il a un attribut obligatoire test dont la valeur est une expression XPath [XPath 99]. L'évaluation de cette expression produit une valeur booléenne. Pour qu'un document soit valide vis-à-vis d'un schéma Schematron, chaque assertion de ce schéma doit être évaluée à vrai. Le contenu de l'élément assert est une phrase déclarative simple exprimant l'assertion en langue naturelle. Lorsque l'attribut test de l'assertion est évalué à faux, une action possible du vérificateur est d'afficher cette phrase en guise de diagnostic ;
- **report** : cet élément est le pendant de l'élément précédent. Cette assertion est vérifiée lorsque son expression est évaluée à faux ;
- **rule** : les deux éléments précédents, assert et report, sont regroupés dans des règles. L'élément rule a un attribut context dont la valeur est une expression XPath. Pour chaque élément du document pour lequel l'expression context est évaluée à vrai, les assertions sont vérifiées. Le contexte d'évaluation de ces assertions est l'élément en question ;
- **pattern** : les règles sont regroupées à l'intérieur de l'élément pattern. Il peut éventuellement avoir un attribut id afin de pouvoir référencer le groupe de règles ;
- **phase** : cet élément permet de spécifier quels groupes de règles sont à vérifier et dans quel ordre. Le contenu de cet élément est une liste ordonnée d'éléments active. Cet élément de type active permet de référencer un groupe de règle grâce à l'attribut pattern qui lui est attaché. Grâce à ce mécanisme de phase, la validation d'un document peut se faire avec plus ou moins de règles ;

- **p** : cet élément correspond à un paragraphe. Il est utilisé pour décrire en langue naturelle le rôle d'un pattern.

La figure 12 donne un exemple d'un schéma Schematron appliqué au type de document filmothèque.

```
1.  <?xml version="1.0" encoding="ISO-8859-1"?>
2.  <schema xmlns="http://www.ascc.nt/xml/schematron">
3.      <title>Schematron Filmothèque</title>
4.
5.      <phase id="Nouveau">
6.          <p>Pour créer un nouveau document</p>
7.          <active id="mini"/>
8.      </phase>
9.
10.         <pattern id="mini">
11.             <rule context="/">
12.                 <assert test="filmothèque">
    Une filmothèque doit avoir un élément filmothèque comme élément racine.
13.                 </assert>
14.                 <report test="count(//filmothèque)>1">
    L'élément filmothèque ne devrait apparaître qu'une seule fois par document.
15.                 </report>
16.             </rule>
17.         …
18.         </pattern>
19.     …
20.     </schema>
```

Figure 12. Fragment du schema Schematron du type de document filmothèque.

Ce schéma définit une phase appelée Nouveau qui active le pattern mini. Ce pattern contient une seule règle pour spécifier qu'un document de type filmothèque doit avoir un élément racine filmothèque, et que cet élément doit apparaître qu'une seule fois dans une instance.

Ce principe de phase est intéressant notamment dans le contexte éditorial. En effet, elle permet aux auteurs d'initier la rédaction de documents avec peu de contraintes. Un des reproches des systèmes d'édition de documents structurés actuels est leur rigidité vis-à-vis du modèle de document sous-jacent. Ce reproche a notamment été formulé par les rédacteurs des manuels aéronautiques. Grâce au mécanisme de contraintes dynamiques, concrétisé par l'élément phase, un système auteur peut suivre le cheminement intellectuel d'un auteur qui généralement part d'une idée plus ou moins structurée pour ensuite l'organiser en adéquation avec son modèle de document.

2.3.4. Synthèse

Les méta-modèles présentés dans cette section permettent de poser des contraintes sur la structure et sur la valeur des attributs. Une deuxième catégorie de méta-langages permet de poser des contraintes sémantiques. Nous avons vu un exemple de ce type de langage à travers RDF schéma [RDFSchema 00]. On peut citer aussi XMI [OMG 00] et UREP [Unisys 01]. L'objectif de ces langages est d'augmenter l'interopérabilité et les échanges entre applications.

Par conséquent, ces langages permettent sans aucun doute d'aller plus loin dans l'automatisation des traitements sur les documents, en particulier pour l'adaptation de document. Cependant, nous avons décidé de ne pas explorer cette voie pour plusieurs raisons. La première raison est qu'il est difficile de concevoir des outils génériques en utilisant cette technologie. Elle repose sur une connaissance très précise du domaine traité. Or dans notre cas, nous nous intéressons à des solutions indépendantes d'un quelconque domaine. Enfin, comme nous l'avons dit ci-dessus, ces langages permettent certainement d'*étendre* la capacité d'adaptation des documents. Ces extensions présupposent donc l'existence de fondations techniques sur lesquels elles reposent. Or ces fondations n'existent pas encore, et c'est sur celles-ci que cette thèse est centrée.

Dans la suite de ce chapitre, nous abordons le problème de la présentation de document structuré.

3. Méthodes et techniques pour la présentation

Un des traitements classiques liés au génie documentaire est l'obtention d'une ou de plusieurs présentations d'un document. Le principe général consiste à associer aux informations logiques contenues dans le document des informations physiques. Selon la nature du document logique et la nature de la présentation, cette association est plus ou moins complexe : plus la distance entre l'information logique et physique est grande et plus le document est complexe à présenter. Par exemple, les documents de type postscript [PostScript 91] contiennent directement des informations graphiques. La présentation est obtenue par l'interprétation de ces informations pour produire une image de qualité.

Puisque l'un des objectifs de ce mémoire est de proposer une solution pour présenter des documents multimédias appartenant à une même classe, nous avons étudié les solutions existantes qui reposent sur l'intégration de la présentation avec la définition du modèle. Nous verrons ensuite le principe bien connu de la séparation du contenu de sa présentation avec les trois processus associé : la transformation, le formatage et la visualisation.

3.1. Intégration de modèles

D'autres domaines que le génie documentaire ont besoin de présenter des documents. En particulier, dans le domaine des bases de données classiques et multimédias, la principale approche est d'étendre les schémas de base de données pour permettre de définir des modèles de présentation [Theodoridis 98][Erwig 99]. En d'autres termes, la présentation est définie dans le modèle lui-même. Une approche pour étendre les schémas de bases de données par des opérateurs temporels est proposée dans [Adiba 95].

Transposé au domaine du génie documentaire, cette approche consiste à étendre un méta-modèle par des constructeurs de présentation. Cette extension doit inclure la définition de média et la composition entre ces médias. Au niveau de la définition des médias, une solution a été expérimentée dans Thot. Le méta-modèle de Thot, langage S [Quint 97], permet de spécifier dans le modèle le type des feuilles de l'arbre selon le média représenté (le média image).

Au niveau de l'organisation, le principe est d'étendre les méta-modèles par des constructeurs autres que purement logiques. En effet, les méta-modèles n'offrent que des constructeurs logiques. Par exemple, les schémas XML offrent les constructeurs de choix, de liste ordonnée

et non ordonnée. Or nous avons vu que de façon générale la description d'un document est organisée en quatre dimensions (cf. section 2.2.1). En particulier les méta-modèles peuvent être étendus par des constructeurs de type séquence et parallèle. Un exemple d'une telle extension est donné figure 13. Cet exemple est un fragment de la définition du modèle de document Docbook. Ce modèle est construit à partir de deux langages existants, XML Schema préfixé par xsd et SMIL préfixé par smil. La définition de l'élément article s'effectue grâce à un opérateur temporel seq. La sémantique de présentation de l'élément seq est étendue pour couvrir la sémantique du constructeur séquence (xsd:sequence) du modèle. Dans ce cas, le seq correspond à une liste ordonnée d'éléments.

```
1.  <xsd:element name="article">
2.      <smil:seq xsd:maxoccurs="unbounded">
3.          <xsd:element ref="section"/>
4.      </smil:seq>
5.  </xsd:element>
6.  <xsd:element name="section">
7.      <smil:par>
8.          <xsd:sequence>
9.              <xsd:element ref="title" maxoccurs="1"/>
10.             …
11.         </xsd:sequence>
12.     </smil:par>
13.  </xsd:element>
```

Figure 13. Exemple de définition d'une classe de document multimédia.

Dans cette approche, pour permettre plusieurs présentations, plusieurs modèles peuvent être spécifiés. Cependant, pour qu'une instance d'un modèle soit valide pour les autres modèles, il est nécessaire de définir les mêmes contraintes logiques pour chaque modèle. Cette condition peut devenir relativement contraignante surtout lorsque les modèles sont très complexes. En outre, c'est une solution peu réaliste car la structure de présentation ne correspond pas toujours à la structure logique du document. Cependant, dans le domaine des bases de données cette solution est intéressante car les besoins sont différents. En particulier, cette approche permet d'exploiter des techniques de recherche d'information portant sur la présentation du document. Les besoins en présentation sont de moindre importance, ainsi que les besoins d'adaptation.

Une seconde approche est de modéliser une classe de document en utilisant un vocabulaire de présentation commun à toutes les classes. Un modèle de document de présentation est alors un patron de ce langage, ce patron étant défini formellement en utilisant un méta-modèle (XML Schema, Schematron, etc.).

La première difficulté de cette approche est de définir, s'il n'existe pas, un méta-modèle très expressif pour pouvoir contraindre de façon syntaxique la complexité d'une présentation multimédia. La seconde difficulté est liée à la charge importante de définition d'un modèle. Car en plus des contraintes de présentation liées au besoin de restitution du domaine (guide touristique, slideshow), il est nécessaire de prendre en compte celles liées au modèle multimédia même (média se joue dans une région, etc.).

De ces deux approches ressort le besoin de séparer les contraintes logiques de celles de nature représentationnelle. Ce besoin est un des piliers du domaine du génie documentaire et a conduit au standard SGML [ISO 86].

3.2. Séparation du contenu de la présentation

Le processus général de présentation d'un document est illustré figure 14. Pour obtenir une présentation d'un document métier, il est nécessaire de lui ajouter des informations de style ou, de façon plus générale, de le *transformer* vers un ensemble d'informations de présentation. Ces informations de présentation sont décrites sous une forme logique, en utilisant un des modèles décrits dans la section 2.2.2 (ou équivalent). Le processus qui permet d'obtenir la forme physique à partir de la forme logique s'appelle le *formatage*. Cette forme physique est ensuite interprétée ou *exécutée* pour produire la présentation finale du document vers le support de sortie.

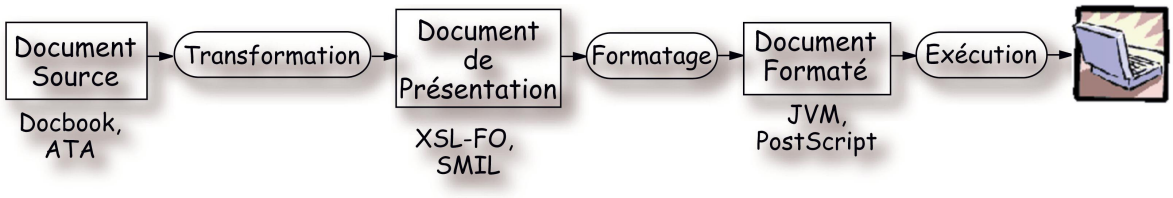


Figure 14. Processus général de présentation.

Il faut noter que l'usage de la transformation dans le contexte de la présentation de document n'est pas encore généralisé. Jusqu'à présent, les applications de présentation utilisent principalement la méthode d'ajout de style, pour les raisons suivantes : d'une part cette méthode est relativement simple à implanter et d'autre part elle permet de concevoir des outils relativement conviviaux pour éditer des documents XML (cf. section 4.2). Cependant, si les styles sont adaptés pour présenter des documents statiques, il n'en est pas de même pour les documents multimédias. Il n'existe pas pour le moment un moyen simple de spécifier des feuilles de styles temporels. De plus, les feuilles de styles ne permettent pas de remettre en cause la structure du document présenté (cf. section 2.2.3.1). Or la structure temporelle et spatiale d'un document multimédia n'est pas toujours la même.

Dans ce mémoire, nous nous focalisons sur le principe de transformation de document que nous décrivons dans la section suivante. Nous verrons ensuite les principes de formatage puis d'exécution.

3.2.1. Transformation

3.2.1.1 Principes de base

Avec l'arrivée des documents structurés est vite apparu un ensemble de besoins relatifs à la transformation [Furuta 88a]. Ces besoins ne sont pas tous dérivés de la problématique de la présentation. La transformation apparaît notamment dans les applications suivantes :

- importation et exportation de documents pour permettre la réutilisation des documents existants ;

- transformation en cours d'édition. L'exemple classique est la transformation d'une liste en un tableau ;
- évolution d'un modèle de document vers une nouvelle version de ce modèle ;
- de façon très générale, conversion de documents entre modèles.

Ce dernier point inclut en particulier la présentation de document. En effet, le principe est de transformer un document, généralement métier, respectant un modèle vers un document de présentation respectant un autre modèle (cf. figure 15). Le document en entrée de la transformation est appelé *document source*. Le document en sortie est appelé *document cible*. Ce type d'application requiert une information additionnelle spécifiant la façon avec laquelle le document doit être transformé. Ces informations sont stockées dans des feuilles de transformation. Cette spécification peut être soit exécutée, soit interprétée par un *processeur de transformation*. Afin d'être réutilisable et générique, c'est-à-dire applicable aux documents appartenant à une même classe, la spécification de la transformation s'appuie sur le modèle du document source. Cependant elle peut aussi s'appuyer sur l'instance source pour réaliser des transformations spécifiques.

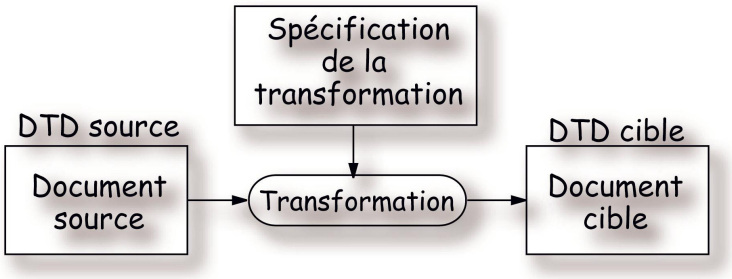


Figure 15. Processus de transformation.

La transformation peut être spécifiée en utilisant un langage de programmation classique. Cependant de nombreuses recherches ont été effectuées afin d'identifier les transformations récurrentes, ainsi que la façon d'interpréter ces transformations. Ces travaux ont conduit à la définition de plusieurs langages dédiés à la transformation de document [Balise, Akpotsui 93, Bonhomme 98, Furuta 88a, Kuikka 95, Mamrak 89, Omnimark5, XSLT 99]. Bien que plusieurs travaux présentés dans ce mémoire soient basés sur le mécanisme de transformation, nous n'effectuons pas un état de l'art sur ce sujet. En effet, un état de l'art complet de ces langages a été accompli dans la thèse de Stéphane Bonhomme [Bonhomme 98]. Un des résultats de cet état de l'art est la classification des langages de transformation selon leur méthode de transformation et de génération. Deux méthodes de transformation ont été identifiées :

- transformations dirigées par la source : la transformation est conduite par un parcours « en profondeur d'abord » du document source. Un ensemble de règles est associé à chaque événement du parcours (début d'un élément, fin d'un élément, etc.) ;
- transformations par requêtes : le document cible est construit par extraction d'informations du document source.

Un des avantages des transformations dirigées par la source est la performance d'exécution. Ce type de transformations requiert un temps d'exécution relativement faible ainsi que peu d'espace mémoire (à nuancer dans certain cas, cf. [Bonhomme 98]). Cependant, elles ne permettent pas de réaliser toutes les catégories de transformations. En particulier, il est impossible de réordonner les éléments dans la structure. Quant à elles, les transformations par requêtes n'ont pas cette limitation mais elles réclament plus de ressources. En particulier, le langage de requête XPath [XPath 99] (cf. section ci-dessous) nécessite que l'arbre source soit entièrement en mémoire.

Nous avons décrit ci-dessus une première approche pour effectuer des transformations. Cette approche, appelée *explicite*, s'appuie sur une spécification externe de la transformation. Une autre approche, appelée *automatique*, consiste à analyser l'information codée dans les modèles de document source et cible pour trouver des relations structurelles entre ces modèles. Ces relations sont ensuite utilisées pour réaliser la transformation. L'approche automatique seule, qui s'appuie sur des relations structurelles, est inexploitable dans les applications, que se soit pour la présentation de document, ou pour l'édition de document [Bonhomme 98]. Les applications de présentation ont besoin en effet de relations plus sémantiques que les règles structurelles utilisées par les transformations automatiques. C'est pourquoi des approches mixtes ont été étudiées, entre autres, dans la thèse de Stéphane Bonhomme. Dans ce mémoire, nous nous positionnons exclusivement dans une approche explicite pour spécifier la transformation. Le langage que nous utilisons est XSLT, que nous décrivons dans la section suivante.

3.2.1.2 Concepts de base de XSLT

XSLT [XSLT 99] est un standard défini par le W3C pour transformer la structure et, dans une moindre mesure, le contenu des documents XML. La spécification de la transformation contient un ensemble de *règles* associées à un *sélecteur*. Une règle permet de spécifier un fragment de document résultat de la transformation pour un ensemble de nœuds source identifiés par le sélecteur. Le corps de chaque règle contient une liste d'*instructions* pour contrôler la transformation et générer le document cible. Les éléments de génération appartiennent soit au modèle XSLT, soit aux modèles cibles. Un exemple va nous permettre d'illustrer ces définitions et de survoler les principes fondamentaux des traitements XSLT. Une description plus technique de ce langage sera donnée dans le chapitre 6.

Le document suivant est une instance de document contenant une liste de mémo. Chaque mémo contient le nom de l'expéditeur, le nom du destinataire, le sujet et le corps du mémo.

```
1.  <?xml version="1.0" encoding="ISO-8859-1"?>
2.  <memoset>
3.      <memo>
4.          <to>Dupont</to>
5.          <from>Dubois</from>
6.          <subject>Réunion</subject>
7.          <body>
8.                  <p>Réunion confirmée <em>demain</em> 10h. </p>
9.          </body>
10.         </memo>
11.
12.         <memo>
```

```
13.             <from>Dupond</from>
14.             <to>Dupont</to>
15.             <subject>Bonne blague</subject>
16.             <body>
17.                 <p>Salut, j'ai trouvé une super image qui va te plaire.</p>
18.                 <p>a+</p>
19.                 <image src="blague.gif"/>
20.             </body>
21.         </memo>
22.     </memoset>
```

Le document suivant est une feuille de transformation XSLT :

```
1.  <?xml version="1.0" encoding="ISO-8859-1"?>
2.  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3.
4.      <xsl:template match="/">
5.          <html>
6.              <body>
7.                  <xsl:apply-templates select="memoset/memo[from='Dubois']"/>
8.              </body>
9.          </html>
10.         </xsl:template>
11.
12.         <xsl:template match="memo">
13.             <p>From: <xsl:value-of select="from"/></p>
14.             <p>To: <xsl:value-of select="to"/></p>
15.             <p><xsl:apply-templates select="body"/></p>
16.         </xsl:template>
17.
18.     </xsl:stylesheet>
```

Cette feuille de transformation est composée de deux règles définies par l'élément template (lignes 26 et 34). La première règle est exécutée, ou *instanciée* lorsque le nœud source en train d'être traité correspond au nœud racine, tandis que la seconde règle est instanciée lorsque le nœud source est de type memo. L'identification des nœuds source pouvant être instanciés pour une règle s'effectue en utilisant un pattern décrit dans la syntaxe XPath [XPath 99]. Par exemple, en XPath, l'identification du nœud racine est représenté par le caractère '/'. Le corps de chaque règle est constitué de deux catégories d'éléments : les éléments XSLT et les éléments appartenant au modèle du document cible. Certains éléments XSLT utilisent le langage XPath pour sélectionner des nœuds sources.

Appliqué au document source ci-dessus, la spécification XSLT ci-dessus permet de générer le document suivant :

```
1.  <html>
2.      <body>
3.          <p>From: Dubois</p>
4.          <p>To: Dupont</p>
5.          Salut, j'ai trouvé une super image qui va te plaire. a+
6.      </body>
7.  </html>
```

L'obtention de ce document est effectué selon le processus de transformation suivant : à l'état initial, une règle de transformation qui correspond le mieux à la racine du document est recherchée. Il faut noter que la racine du document n'est pas memoset mais le nœud parent de memoset. Dans notre exemple, c'est la règle ligne 4 qui est trouvée. Ensuite l'exécution de la règle s'effectue de façon séquentielle. Les éléments qui n'appartiennent pas au langage XSLT sont directement copiés dans le document cible. Dans notre exemple, l'élément html, puis l'élément body, sont successivement copiés. Les éléments de transformation ont une sémantique différente selon leur type. Voici quelques éléments XSLT avec la description de leur sémantique :

- apply-templates : lorsque cet élément est exécuté, une liste ordonnée de nœuds source est sélectionnée à partir de la requête spécifiée avec l'élément sous forme d'attribut (attribut select). Pour chacun de ces nœuds source, une règle est recherchée et appliquée. Par exemple, l'instruction ligne 29 sélectionne l'ensemble des nœuds memo dans memoset dont l'élément enfant from contient la chaîne de caractères Dubois. Cette expression a pour effet de sélectionner le nœud memo ligne 3. Une règle est ensuite recherchée pour ce noeud, celle de la ligne 34 est trouvée et est exécutée ;

- value-of : cette instruction génère du texte à partir du contenu source sélectionné. Par exemple <value-of select="from"/> génère le texte correspondant au contenu de l'élément from ;

- if : cette instruction a un attribut test qui spécifie une expression booléenne. Si l'évaluation de cette expression est vraie, alors le contenu de l'instruction if est instancié, sinon rien n'est créé ;

- include/import : ces instructions permettent respectivement d'inclure ou d'importer un ensemble de règles spécifié dans un fichier externe. L'inclusion des règles de transformation est un simple ajout de règles qui complète la feuille de transformation. L'importation de règles définit une hiérarchie de feuilles qui permet de compléter et surcharger les règles de la feuille importée. Grâce à ces instructions, il est possible de modulariser les feuilles de transformation. Il est possible par exemple de séparer les différentes dimensions caractérisant un document multimédia. De plus, les règles s'appliquant à la DTD du document sont séparées de celles spécifiques à l'instance.

En positionnant XSLT par rapport aux méthodes de transformations énoncées dans la section précédente, ce langage permet une transformation à la fois dirigée par la source et par requête. En effet, le choix d'ordre d'application des règles est guidé par le document source tandis que la génération du document cible s'effectue en extrayant les données de la source.

3.2.2. Formatage

L'opération qui consiste à convertir des informations de présentation logique vers des informations concrètes est appelée *formatage* [Furuta 82, Quint 87]. Un des premiers exemples de formatage, cité dans [Furuta 82], est la transformation de caractères vers leur représentation dans une police de caractères particulière, la production de mot en deux dimensions en tenant compte d'une éventuelle coupure du mot logique, etc. Avec l'arrivée des documents multimédia, le formatage est devenu plus complexe pour prendre en compte les multiples dimensions. On parle alors de formatage spatial et temporel.

De nombreux travaux portent sur les techniques de formatage, les premiers datant des années 60. Les principaux points à retenir sont les suivants :

- Le formatage peut être plus ou moins complexe selon le niveau des informations à formater. Trois niveaux sont identifiés :
    - le calcul de valeurs absolues. C'est le cas le plus simple qui consiste par exemple à convertir une valeur exprimée dans une certaine unité vers une autre unité (le pixel) ;
    - le calcul hiérarchique sans cycle. Par exemple, le calcul de valeurs héritées correspond à un calcul hiérarchique sans cycle ;
    - le calcul hiérarchique avec cycle. Par exemple, le formatage des documents de type SMIL1.0 requiert des algorithmes très simples [Navarro 01]. Par contre, le langage Madeus, qui permet de spécifier des relations spatiales et temporelles de haut niveau, nécessite l'utilisation de résolveurs de contraintes. Ces résolveurs sont complexes à mettre en œuvre et nécessitent de nombreuses ressources en temps d'exécution et en place mémoire ;
- les besoins lors de la conception d'algorithme de formatage sont différents selon que le formatage est intégré dans un éditeur ou non. Les systèmes de formatage non intégrés, comme TEX [Knuth 84], privilégient généralement la qualité du formatage. Par contre, pour avoir un éditeur réactif [Thot], le système de formatage intégré doit être très performant. Pour cela des techniques incrémentales sont utilisées (cf. section 4.2.5) ;
- comme le document peut être visualisé (et/ou écouté) sur différents périphériques de sortie, le formatage doit tenir compte des spécificités de son support final : écran, imprimante, haut-parleur, vidéo projecteur, etc. Par exemple, la conversion d'une longueur exprimée en centimètre dépend de la résolution de l'écran ou de l'imprimante utilisé comme support de sortie. De plus, comme un écran a une résolution plus faible qu'une imprimante, le formatage peut être moins précis et donc plus rapide ;
- par rapport au processus de transformation, le formatage s'appuie sur un langage source unique et généralement bien défini. Cela permet de développer des algorithmes optimisés pour formater rapidement un document.

Le résultat du formatage consiste en un ensemble d'informations interprétables directement par le processus d'exécution décrit dans la section suivante.

3.2.3. Visualisation et exécution

La dernière étape du processus de présentation consiste à visualiser et/ou à écouter les informations concrètes résultantes du formatage. Cette étape s'effectue pour un périphérique de sortie donné. Pour les documents statiques, l'opération de visualisation est relativement simple car les informations de formatage sont directement interprétables par les pilotes d'affichage graphique. Par contre, lorsque le document est de nature multimédia, le résultat du formatage produit une structure temporelle. Cette structure est interprétée par une machine d'exécution qui permet de visualiser temporellement le document [Pérez 96, Song 96, Layaïda 97, Sabry 99].

4. Méthodes et techniques pour l'édition

L'édition est la phase pendant laquelle les structures et le contenu du document sont créés et mis à jour par un auteur. Dans le but d'augmenter la productivité de l'auteur, un certains nombre de services lui sont offerts. Ces services sont regroupés dans un *système d'édition*. De façon générale, l'objectif du système d'édition est d'assister l'auteur pendant les phases du cycle d'édition d'un document : la création, l'édition, la publication, la maintenance (ou réédition) et le stockage. Pour cela, le système d'édition affiche plus ou moins graphiquement le document et offre plusieurs fonctions liées au cycle d'édition. Ces fonctions sont de natures diverses, comme l'ouverture d'un fichier, l'ajout et la suppression de contenu, la modification du style, etc. Plusieurs approches sont recensées parmi les outils d'édition existants. Nous proposons dans cette section un tour d'horizon de ces différentes approches. La première section définit un ensemble de besoins relatifs à la conception de systèmes d'édition. Ces besoins sont exprimés à travers plusieurs critères de qualité que nous définissons. Dans la seconde section, nous analysons les systèmes d'édition existants en insistant sur l'aspect fonctionnel de ces outils.

4.1. Critères de qualité

La qualité du système d'édition varie selon le respect ou non d'un certain nombre de critères (critères de qualité). Nous avons identifié deux groupes de critères, le premier étant les critères *fonctionnels* et le second les critères *ergonomiques*. Les critères fonctionnels que nous avons retenus sont les suivants :

- le système d'édition doit couvrir l'ensemble du *pouvoir d'expression* du modèle de document sous-jacent ;
- le processus d'édition doit être *incrémental*. Puisque le processus d'édition est interactif, il est essentiel de réduire le temps de réponse après une action de l'utilisateur [Coutaz 90]. On cherchera donc à recalculer uniquement les données nécessaires ;
- *l'intégrité* du document doit être maintenu. Le système d'édition doit s'assurer que le document est valide par rapport à un modèle. Cette validation peut intervenir au moment du stockage du document, ou bien en continu pendant l'édition.

Puisqu'un système d'édition s'adresse à un utilisateur, il doit respecter un certain nombre de critères ergonomiques. De nombreuses méthodes ont été définies par la communauté « interaction homme machine » afin de concevoir et d'évaluer les interfaces utilisateurs [Balbo 94]. Ces méthodes reposent sur des critères ergonomiques plus ou moins formels. Notre objectif dans ce mémoire n'est pas d'appliquer une méthode formelle donnée car elles sont longues et coûteuses. De plus, ce mémoire porte principalement sur l'aspect fonctionnel plutôt que sur l'aspect ergonomique. Néanmoins, nous serons amené à effectuer des choix d'interface utilisateur que nous tenons à argumenter selon des critères informels mais reconnus pour leur pertinence. Ces exigences nous ont conduit à choisir les critères ergonomiques définis par Bastien et Scapin [Bastien 93, Smith 86]. Voici un extrait de ces critères :

- la **charge de travail** permet de réduire la charge mnésique (mémoire à court terme). Elle concerne l'ensemble des éléments de l'interface qui ont un rôle, pour l'utilisateur,

dans la réduction de sa charge perceptive ou mnésique et dans l'augmentation de l'efficacité du dialogue ;

- l'**adaptabilité** propose différents niveaux d'utilisation du logiciel. Elle se réfère à la capacité du système à réagir selon le contexte et selon les besoins et préférences de l'utilisateur ;

- la **gestion des erreurs** doit réduire les occasions d'erreur et toute erreur doit être détectable dès son occurrence et doit pouvoir être corrigée. Elle concerne tous les moyens permettant d'une part d'éviter ou de réduire les erreurs, et d'autre part de les corriger lorsqu'elles surviennent ;

- l'**homogénéité** s'apparente à la notion de cohérence (par exemple, séquence de commandes identiques pour un même résultat). Elle se réfère à la façon avec laquelle des choix d'objets de l'interface (code, procédures, dénominations, etc.) sont conservés pour des contextes identiques, et des objets différents pour des contextes différents. L'homogénéité s'applique aussi bien à la localisation et au format qu'à la syntaxe et à la dénomination ;

- la **compatibilité** suppose un faible recodage des informations entre le savoir de l'utilisateur et le format imposé par le logiciel. Elle se réfère à l'accord pouvant exister entre les caractéristiques de l'utilisateur (mémoire, perceptions, habitudes, etc.) et l'organisation des sorties, des entrées et du dialogue.

Maintenant que nous avons déterminé un certain nombre de critères de qualité, nous étudions les approches existantes pour éditer les documents structurés.

4.2. Systèmes d'édition existants

En analysant les systèmes d'édition existants, il apparaît clairement trois catégories de systèmes auteur. La première catégorie regroupe les systèmes auteur dédiés à un ou plusieurs modèles de document structuré. Ces systèmes connaissent *a priori* le ou les modèles de document qu'ils sont capables de manipuler. La seconde catégorie englobe les éditeurs « génériques » : ils permettent d'éditer tous les modèles de document structuré. Enfin les systèmes d'édition appartenant à la troisième catégorie sont un mélange des deux approches précédentes.

Nous commençons par décrire les outils appartenant à la première catégorie. Une partie de ces outils sont dédiés au langage XSLT. Comme l'édition de ce modèle constitue une part importante du chapitre 5, nous détaillons plus particulièrement ces outils dans une deuxième sous-section. Ensuite nous décrivons successivement les outils appartenant à la seconde catégorie puis à la troisième catégorie.

4.2.1. Outils dédiés à un modèle de document

Les systèmes d'édition de type dédié sont conçus pour éditer un ou plusieurs modèles de document bien précis. Les exemples de systèmes d'édition de ce type sont nombreux dans l'industrie. Nous pouvons citer par exemple Amaya [Amaya] et Macromedia Dreamweaver [Dreamweaver 4] pour les documents HTML, Grins [GRiNS] et LimSee [LimSee] pour les documents SMIL, Jasc WebDraw [WebDraw] pour les documents SVG, etc. Tous ces outils répondent aux mêmes exigences suivantes :

- puisque leur conception s'appuie sur un ou plusieurs modèles particuliers, les fonctionnalités fournies sont en adéquation avec le ou les modèles sous-jacents. Par conséquent, ce type d'approche permet de répondre au critère du pouvoir d'expression ;

- comme ces outils connaissent la sémantique du modèle, des actions d'édition de haut niveau peuvent être offertes. Par exemple, Macromedia Dreamweaver propose de valider les liens de navigation ; Visual XSLT permet de tester des documents XSLT ;

- ils conservent l'intégrité du document. Généralement, ces outils permettent uniquement des actions valides par rapport au modèle de document. Par conséquent, le document est valide tout au long de la phase d'édition. Cependant, lorsque le système d'édition permet d'éditer le document à travers une vue source, il peut devenir incohérent. Dans ce cas, le système détecte ce ou ces incohérences en utilisant des outils de validation de document par rapport à un modèle.

Cette approche permet de créer des éditeurs de qualité qui répondent aux critères définis dans la section précédente. Cependant la conception de tels outils est longue et très coûteuse. Afin de palier à ce problème, des solutions pour faciliter la création de tels environnements ont été étudiées. Souvent, ces solutions se concrétisent par la définition de boîtes à outils (DirectShow, JMF[11], Kaomi [Tardif 00], MAVA [Hauser 00], GTK[12]).

On peut noter que la plupart des modèles de document sont spécifiés sans considérer les applications de type éditorial. Or, les besoins de modélisation peuvent varier sensiblement pour offrir des fonctions d'édition de haut niveau. Par exemple, les outils comme SmilEditor permettent de spécifier et de maintenir des relations spatiales entre les objets du document (aligner à gauche, espacer de, etc.). Ce type de spécification n'est généralement pas représenté par le modèle, en l'occurrence SMIL. Rares sont les outils qui étendent ces modèles pour stocker les informations d'édition.

4.2.2. Outils dédiés au modèle de document XSLT

Dans la suite de ce mémoire nous nous intéresserons particulièrement à l'édition de feuilles de transformation XSLT. C'est pourquoi nous proposons de faire un tour d'horizon des outils d'édition existants qui répondent à ce besoin.

Nous avons identifié deux catégories d'outils : les outils génériques et les outils spécifiques. L'objectif des outils d'édition appartenant à la première catégorie est de créer n'importe quel type de transformation. La plupart de ces outils sont équivalents à des environnements de développement [VisualXSLT, XSLDebug, StylusStudio]. Ils fournissent une vue textuelle du code XSLT et permettent de tester l'exécution de la transformation. Lorsque le document cible de la transformation est un document HTML, ces outils proposent généralement une prévisualisation du résultat. Aucune action n'est possible sur ce résultat, à part vérifier qu'il correspond aux attentes de l'auteur. Une autre approche, proposée par Emmanuel Pietriga [Pietriga 01], est basée sur la programmation visuelle. Une représentation graphique pour visualiser et manipuler les transformations de documents est associée à chaque concept relatif à XML et à XSLT. Par exemple, la figure 16 montre à gauche le document source constitué

[11] Java Media Framework, http://java.sun.com/products/java-media/jmf/index.html.

[12] The GIMP Toolkit, http://www.gtk.org/.

d'un élément de type subject représenté par un carré et son contenu textuel représenté par un losange. La partie droite est une représentation graphique de la transformation qui, dans ce cas, génère un élément de type subject contenant deux nœuds textuels. Le premier nœud possède une valeur fixe (Re :), tandis que la valeur du second nœud est copiée à partir du document source. L'édition d'une telle transformation s'effectue à travers une interface graphique entièrement « zoomable ».

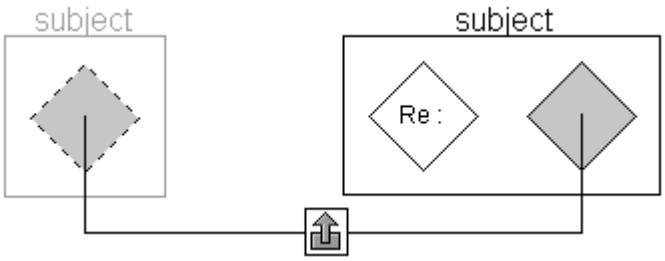


Figure 16. Approche visuelle pour l'édition de transformations.

Les outils d'édition qui appartiennent à la seconde catégorie permettent l'édition de transformations pour générer exclusivement des présentations HTML ou XHTML. Par conséquent, ces outils peuvent proposer des fonctions d'édition de haut-niveau relatives à l'édition de présentations. Un seul outil, à notre connaissance, entre dans cette catégorie : <xslComposer> [<xsl>Composer]. Il permet l'édition de feuilles de transformation XSLT par manipulation directe. La fenêtre principale est composée de quatre parties. La première partie montre le document XML en cours d'édition sous la forme hiérarchique. La seconde partie affiche la liste des règles de transformation. La troisième partie montre les règles qui sont appliquées et la dernière partie montre la présentation finale et le code XSLT. La création de nouvelles règles s'effectue par une action de glisser-lâcher d'éléments XML vers la présentation finale. L'édition du contenu des règles, exprimé en HTML, est effectuée dans une vue WYSIWYG séparée. Bien que cet outil permette l'édition des transformations XSLT de façon interactive, il le fait de façon limitée. Un sous ensemble du langage XSLT est pris en compte. En particulier, cet outil ne permet par d'imbriquer l'application de règles : les règles sont appliquées en séquence uniquement. Par conséquent, il est impossible de créer des présentations pour des documents récursifs. De plus, la transformation est exécutée depuis le début après chaque modification de la feuille de transformation. Par conséquent, plus le document est de taille importante et plus la génération du résultat sera longue.

4.2.3. Outils indépendants du modèle

La façon la plus naturelle d'éditer un document est de le faire à travers une représentation textuelle ou au mieux à travers une représentation sous forme d'arbre. Les caractéristiques de ces outils [XMLSpy, Merlot] sont les suivantes :

- puisque le système d'édition ne connaît pas la sémantique du document, seules des actions de bas niveau sont offertes pour manipuler la structure et le contenu. Ces actions sont par exemple l'ajout d'un élément, le déplacement d'un sous arbre, etc. ;
- pour vérifier l'intégrité du document, deux familles d'outils existent [Rekik 01] :

- ceux qui contraignent l'auteur à respecter le modèle à tout moment de l'édition ;
- et ceux qui ne contrôlent pas en permanence la validité du document. Ce contrôle s'effectue soit à la demande de l'auteur, soit lors de la sauvegarde.

Ce type d'outils permet d'éditer n'importe quel modèle de document. Cependant, puisque seules des fonctions d'édition de bas niveau son offertes, la tâche d'édition d'un document est fastidieuse, mais surtout non productive. Il est nécessaire d'interpréter (de visualiser) le document après l'édition pour vérifier que le résultat correspond aux attentes de l'auteur.

4.2.4. Outils mixtes

L'édition de documents génériques est une tâche nécessitant un effort cognitif important. Pour diminuer cet effort, l'idée est d'utiliser un langage de présentation. Le système permet d'éditer des informations de présentation et de les associer aux éléments du document. La plupart de ces systèmes existants reposent sur le langage de style CSS [XMetal2.0]. L'utilisation de ce langage simplifie l'édition du document car le contenu du document est accessible graphiquement. Cependant, le pouvoir d'expression des feuilles de style reste limité. Elles ne permettent pas de spécifier des présentations complexes et d'étendre les fonctionnalités du système d'édition avec, en particulier, de nouvelles actions d'édition.

Pour palier à ce problème, certains outils utilisent le mécanisme de transformation de document. Par exemple, Thot [Thot] permet d'éditer le contenu d'un document à travers une présentation générée par transformation. Cependant, le langage de transformation ad hoc utilisé (le langage T [Quint 97]) repose sur l'approche des transformations dirigées par la source (cf. section 3.2.1.1). Par conséquent les présentations générées sont très proches de la structure du document métier.

L'intérêt d'utiliser des outils mixtes se limite aux documents métier. En effet, pour les documents de présentation, le langage de présentation doit être aussi expressif que le langage de présentation inclus dans l'outil. Par conséquent, autant développer un outil dédié à ce langage de présentation.

4.2.5. Bilan sur les outils d'édition

Les outils dédiés et mixtes présentés dans cette section reposent sur l'architecture illustrée figure 17. Le document édité est d'abord présenté à l'auteur en utilisant un processus de présentation comme celui décrit dans la section 3. Lorsque l'auteur modifie son document, il peut agir soit directement sur la visualisation du document, soit le faire indirectement, par exemple en utilisant un menu. Dans le premier cas, le système d'édition doit être capable, à partir d'une action sur le document formaté de répercuter l'action sur le document édité (formatage inverse).

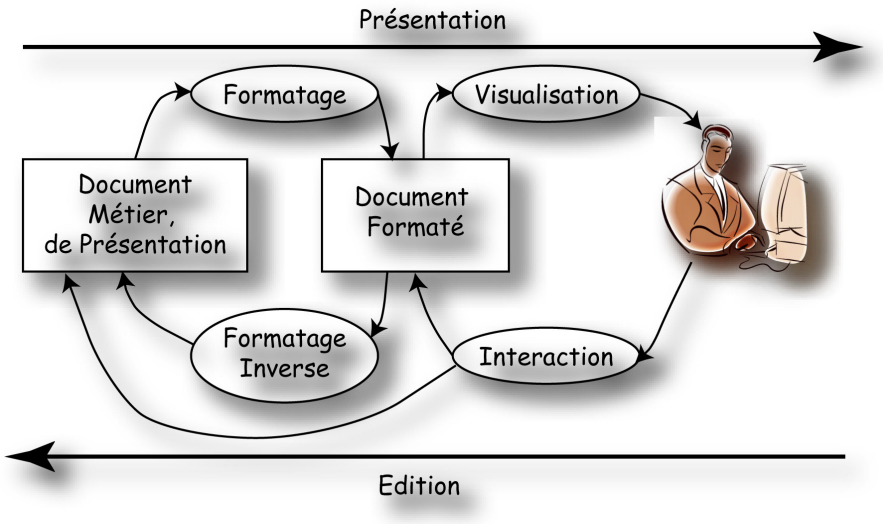


Figure 17. Processus d'édition de documents XML.

4.3. Algorithmes incrémentaux

Une opération d'édition entraîne généralement une légère modification du document. Au lieu réappliquer complètement le processus de présentation, seule la portion affectée doit être refaite. Par souci d'efficacité, l'utilisation d'algorithmes incrémentaux s'avère indispensable. C'est pourquoi plusieurs algorithmes incrémentaux ont été conçus autour de l'édition documentaire [Furuta 88b, Lindén 94]. La plupart du temps, ces algorithmes utilisent une méthode ad hoc. Or, la conception et l'utilisation de tels algorithmes ne sont pas réservées à l'édition documentaire [Ramalingam 93]. Les algorithmes incrémentaux sont utilisés également dans les résolveurs de contraintes, pour la compilation de programme et bien d'autres domaines. C'est à partir de l'expérience acquise lors du développement de tels algorithmes que plusieurs techniques génériques ont été clairement identifiées. Comme nous serons amenés à développer des algorithmes incrémentaux, nous décrivons dans la suite de cette section les techniques rencontrées fréquemment dans la littérature.

Avant de décrire ces techniques, nous donnons une définition formelle du problème de « rendre un programme incrémental » (cf. figure 18) [Ramalingam 93] :

> *Étant donnés un programme f et une opération ⊕, un programme f' est appelé une version incrémentale de f par rapport à ⊕ si f' calcule f(x ⊕ Δx) de façon efficace en utilisant f(x).*

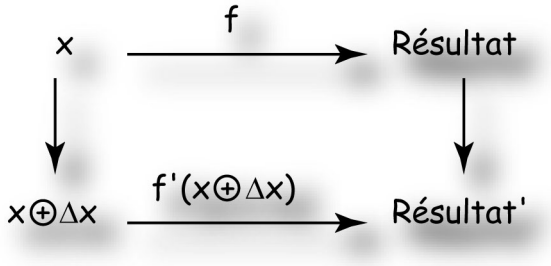


Figure 18. Calcul incrémental.

Par exemple, supposons que *f* soit un compilateur C, *x* un programme C et que l'opération $\oplus$ effectue un changement du programme C. *f'* est alors un compilateur de programme C incrémental : ce compilateur compile en effet un nouveau programme en mettant à jour l'ancien programme plutôt que de le compiler dans sa totalité.

Les techniques permettant de dériver un programme incrémental à partir d'un programme non incrémental sont, entre autres, les suivantes [Liu 95, Pugh 87, Sundaresh 91] :

- **exploitation du résultat précédent :** le calcul de $f'(x \oplus \Delta x)$ s'effectue en utilisant le résultat du calcul précédent $f(x)$. Pour cela il est nécessaire de stocker le résultat $f(x)$ dans un cache ;

- **exploitation de résultats intermédiaires :** de façon générale, pour calculer $f(x)$ plusieurs calculs intermédiaires sont nécessaires. Ces calculs peuvent aussi être stockés pour calculer $f'(x \oplus \Delta x)$;

- **évaluation partielle :** cette technique s'appuie sur une partie connue et statique de la donnée en entrée. Dans certain cas, la fonction peut être simplifiée en évaluant de façon statique cette partie.

On peut noter que l'utilisation d'algorithmes incrémentaux au sein du domaine du génie documentaire n'est pas réservée à l'édition de document. Ils sont utilisés aussi pour obtenir une évaluation partielle d'une présentation, ou encore lorsqu'un document est reçu en utilisant un flux de données (*streaming*). Dans la suite de ce mémoire, nous verrons une utilisation concrète de ces techniques dans le cadre de l'édition.

5. Conclusion

Dans ce chapitre nous avons passé en revue plusieurs travaux relatifs au génie documentaire : la modélisation, la présentation et l'édition de documents structurés. De cette étude, il ressort les lacunes suivantes :

- pour le moment, il n'existe pas de système permettant de présenter des documents multimédia appartenant à une classe donnée. Les feuilles de style sont insuffisantes lorsque plusieurs dimensions caractérisent un document ;

- les méta-modèles sont encore très peu exploités en dehors du contexte de la validation de document ;

- les modèles de documents et les techniques pour présenter ces documents ne prennent pas en compte les contraintes du matériel utilisé comme support de visualisation.

Ce dernier point nécessite une étude plus approfondie sur le problème de l'adaptation de document. Ce problème fait l'objet du chapitre suivant.

Chapitre 3

Adaptation de documents multimédias

Dans le chapitre précédent, nous avons effectué un tour d'horizon sur les différents modèles de présentation existants ainsi que sur les diverses techniques pour les interpréter. Nous avons montré les limites des outils basés sur ces modèles et ces techniques notamment au niveau de leur incapacité à s'adapter au contexte de l'utilisateur. Or ce besoin d'adaptation se fait de plus en plus pressant avec l'apparition ces dernières années d'une multitude de nouveaux matériels pour consulter de l'information.

L'objectif de ce chapitre est d'étudier les principes émergeants relatifs à ce problème. Pour cela, nous commençons par définir le problème d'adaptation notamment en identifiant les composants d'un système adaptable. Ensuite nous étudions les deux composants maîtres d'un système adaptable : le contexte de l'utilisateur et le système d'informations. Pour finir, nous décrivons les systèmes adaptables existants et montrons les limites de ceux-ci.

1. Introduction

1.1. Définitions

Les composants qui interviennent pour résoudre le problème de l'adaptation constituent un *système adaptable*. La figure 19 illustre ces composants indépendamment de leur organisation physique (client, serveur, proxy, etc.). La fonction principale d'un tel système est de restituer de l'information regroupé dans un *système d'informations*, tout en satisfaisant aux besoins et aux contraintes définis par le *contexte de l'utilisateur*.

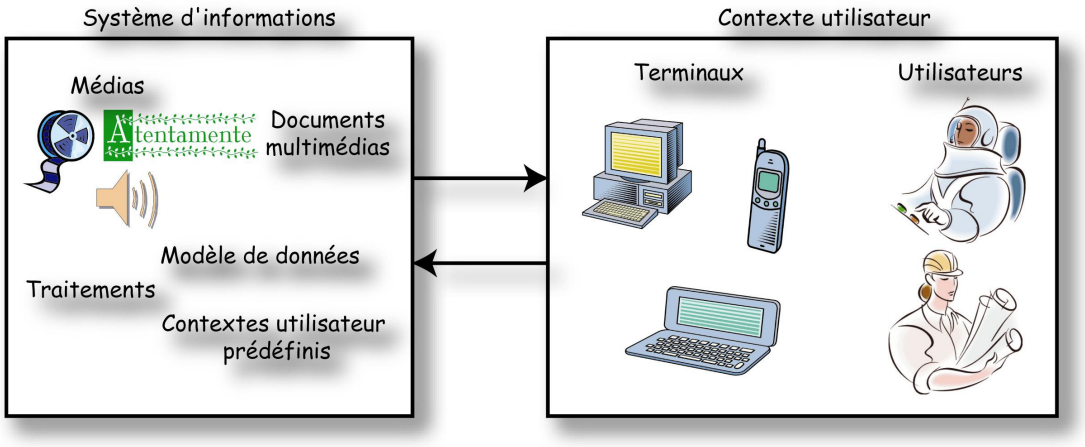


Figure 19. Principaux composants d'un système adaptable multimédia

De façon générale, le contexte utilisateur est caractérisé par :

- le *terminal* d'accès à l'information, par exemple une station de travail, un assistant personnel, etc.

- *l'utilisateur*, c'est-à-dire une personne utilisant un terminal et accédant à de l'information, soit localement ou soit par l'intermédiaire d'un réseau. Aucune restriction n'est, *à priori*, posée sur les capacités de l'utilisateur (physiques, contraintes extérieures, cognitives, etc.).

Le système d'information est, dans notre cas, constitué d'un ensemble de documents multimédias, de médias, de méta-données, de divers traitements et éventuellement de plusieurs contextes utilisateur prédéfinis. La communication bilatérale entre le système d'informations et le contexte utilisateur peut s'effectuer en utilisant plusieurs types de moyens d'accès. Cela peut être une communication locale via le système de fichiers ou un accès réseau comme l'Internet.

La fonction principale du système adaptable, qui le différencie d'un système de présentation classique, est de confronter le système d'informations auquel un utilisateur accède avec le contexte l'environnant. Le résultat de la confrontation, c'est-à-dire l'adaptation du système d'informations au contexte de l'utilisateur, est la production d'une présentation qui correspond aux besoins et aux contraintes du contexte de l'utilisateur. Par exemple, pour un utilisateur malvoyant, une adaptation réalisée par le système adaptable est la production d'objets de nature sonore de façon à faire entendre un texte plutôt l'afficher.

Depuis longtemps, la recherche s'intéresse à la création de systèmes qui soient accessibles par le plus grand nombre de personnes. Il a été rapidement établi que ce n'était pas à l'utilisateur de s'adapter au système, mais au système de s'adapter à l'utilisateur. Cependant, la conception de tels systèmes est complexe en particulier à cause de la diversité des utilisateurs. Cette diversité se situe à de multiples niveaux : physique, culturel, intellectuel ou encore émotionnel. À la base de l'adaptation se trouvent donc les premiers travaux concernant la *modélisation de l'utilisateur*. Ces travaux ont été initiés dans le domaine de l'intelligence artificielle, notamment par Allen [Allen 90], Kobsa [Kobsa 93] et Rich [Rich 79]. Par exemple, l'objectif du système GRUNDY décrit dans [Rich 79] est de se mettre à la place d'un libraire pour recommander des livres. Ces recommandations se fondent sur les caractéristiques de l'utilisateur telles que ses préférences sur la catégorie des livres, sa tolérance vis-à-vis de la sexualité, la violence, la souffrance, etc. Nous verrons plus en détails les différents attributs du modèle de l'utilisateur dans la section 2.1.

1.2. Évolution du matériel

Après le besoin d'adapter un système au modèle de l'utilisateur, un nouveau besoin est apparu avec l'arrivée ces dernières années, d'une multitude de nouveaux terminaux. Comme en témoigne le tableau 1, ces terminaux sont caractérisés par des possibilités très variées, que ce soit physiques mais aussi logicielles. Par exemple, le téléphone cellulaire, comme le Nokia 7110, est caractérisé par un écran de petite taille, non graphique, et par un réseau à faible débit. Le téléphone cellulaire Motorola P1088 est, quant à lui, caractérisé par un écran plus large mais toujours avec un réseau à faible débit. Le Nokia 7110 ne permet pas de cliquer sur les images, ce que permettent certains téléphones WAP. De façon générale, les assistants personnels comme l'iPaq

Figure 20. Montre fonctionnant sous le système d'exploitation Linux.

offrent un plus grand espace de travail que les téléphones cellulaires. Ils sont cependant limités en ressources mémoire et de calcul par rapport à des ordinateurs portables ou des stations de travail. Plus récemment, c'est avec les objets de la vie quotidienne que nous communiquons : vélo interactif, screen fridge [Screenfridge], e-buro, écharpe communicante [Echarpe], Web parfumé, montre (cf. figure 20), etc.

Terminaux	Écran	Mémoire Maximum	Réseau	Types de média supportés	Date de mise en service
Montre (Montre IBM (cf. figure 20))	Graphique (résolution non connue), monochrome	8MB	IrDA, Radio Frequency wireless connectivity	Texte, Image	Prototype 2001
Téléphone cellulaire WAP (Nokia 7110)	96x65 pixels, monochrome	Inconnue	GSM (9.6 Kbps), HSCSD[13] (56 Kbps)	Texte (sans style), Audio (voix), Image (WBMP)	Octobre 1999
Téléphone cellulaire WAP/HTML (MotorolaP1088)	192/272 pixels, 16 niveaux de gris	2 Mo	GSM (9.6 Kbps)	Texte (avec style), Audio (voix), Image (JPEG, GIF)	Février 2000
PDA Pocket PC (iPaq H3600)	240x320 pixels, 4096 couleurs	64 Mo	Ethernet 802.11 (11 Mps), 802.3x (100MBps)	Texte (avec style), Audio, Image, Vidéo	Juin 2000
Portable (Dell) (Inspiron 8100)	1400x1050 pixels, 16 millions de couleur	512 Mo	Ethernet 802.11 (11 Mps), 802.3x (100MBps)	Tous types de média	2001
Station de travail (IBM IntelliStation E Pro6836/6846)	1600x1280, 42 millions de couleur (32 plans)	1,5 Go	Ethernet 802.11 (11 Mps), 802.3x (100MBps)	Tous types de média	Juin 2001

Tableau 1. Quelques caractéristiques de plusieurs terminaux.

Avec une telle disparité dans les capacités de ces objets communicants, on conçoit aisément qu'une application documentaire conçue pour une station de travail ne convienne pas pour un

[13] High Speed Circuit Switcher Data

assistant personnel ou un téléphone cellulaire. Il convient donc de prendre en compte cette diversité en commençant par *caractériser les terminaux* de façon plus rigoureuse, ce que nous faisons dans la section 2.2.

1.3. Génération versus adaptation

La modélisation de l'utilisateur et la caractérisation des terminaux constituent un ensemble de *paramètres d'adaptation*. Le type d'adaptation qui réalise et contrôle la confrontation entre ces paramètres et les possibilités du système d'informations peut être très varié. De façon générale, deux types d'adaptations sont distingués : la génération de contenu et l'adaptation de la présentation. Le premier type fait référence aux techniques de *génération de contenu* adapté au contexte de l'utilisateur. Par exemple, lorsqu'un élève consulte un cours de mathématiques sur la résolution d'équations du second degré, une adaptation possible est, pour un élève débutant, d'ajouter des exercices sur la résolution d'équations du premier degré. Pour un élève plus expérimenté, seuls les exercices sur la résolution d'équations du second degré sont donnés. Dans cet exemple, le contenu même du document change. On peut noter que de nombreux travaux traitent du problème de la génération de contenu, en particulier dans les domaines de la linguistique, de l'intelligence artificielle [Kantrowitz 93, Reiter 00, Kobsa 01b] et du multimédia [Buchanan 95, Dalal 96].

Dans ce mémoire, nous nous focalisons sur l'*adaptation de la présentation* et en particulier des présentations multimédias. À l'inverse de la génération de contenu, un des impératifs de l'adaptation d'une présentation est de conserver le plus possible le contenu véhiculé à travers cette présentation. En reprenant l'exemple d'un cours interactif, une adaptation possible au niveau de la présentation est de montrer, pour un élève réfléchi, les cours théoriques avant les exercices. Pour un élève impulsif, les exercices sont montrés avant les cours théoriques. Seule la façon de véhiculer ce contenu peut être amené à changer, par exemple en changeant de type de média. Cependant, ce changement doit suivre des règles précises, à commencer par le respect des intentions de l'auteur. Une adaptation qui respecte les intentions de l'auteur est appelée *adaptation conforme* ou monotone. Lorsque l'adaptation ne peut pas se faire dans ces conditions, alors on parle d'*adaptation par transgression* [Diaz 01]. Dans ce cas, les intentions de l'auteur doivent être interprétées de façon plus lâche pour permettre un certain degré d'adaptation, tout en essayant de minimiser la transgression.

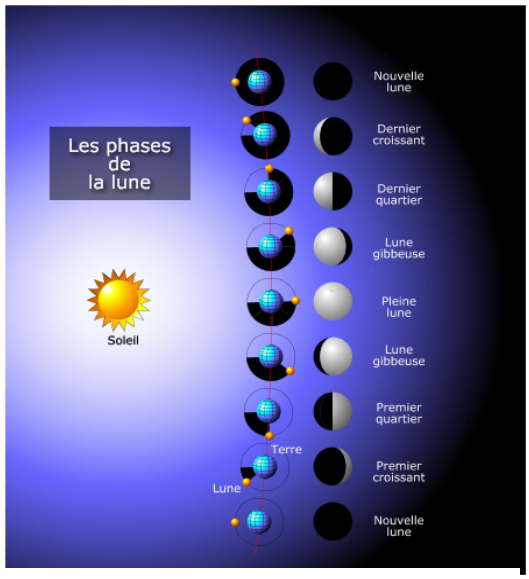


Figure 21. Dans ce schéma, la position du soleil fait partie de la sémantique du document.

Dans les deux cas, l'adaptation doit se faire en respectant un certain nombre de critères. En particulier, le résultat de l'adaptation doit produire une présentation qui est utilisable, cohérente et de bonne qualité [Ivory 01, Mendes 01]. De plus, comme nous l'avons dit

précédemment, elle doit conserver dans la mesure du possible le contenu initial de la présentation en sachant que ce contenu est véhiculé par les médias eux mêmes, mais aussi par leur organisation. Par exemple, un schéma montrant les phases de la lune (cf. ci-contre) ne peut être bien compris que si le dessin du soleil est à gauche des représentations des phases de la lune.

1.4. Exemples

Nous illustrons ci-dessous la diversité des adaptations possibles à travers plusieurs exemples. Toutes les adaptations que nous présentons ci-dessous se situent au niveau de la présentation, cible de nos travaux. Pour chacun des exemples, nous mettons en évidence le ou les paramètres d'adaptation impliqués ainsi que la ou les solutions possibles au niveau de la présentation.

- Pour les personnes aveugles ou malvoyantes, qui accèdent à l'information à partir d'une station de travail (cf. tableau 1), la modalité de présentation de l'information doit être modifiée en une modalité tactile et/ou auditive. Il est donc nécessaire de pouvoir transformer des objets de différentes natures (texte, vidéo, animation, etc.) en objets équivalents sonores ou tactiles [Mynatt 94]. Par exemple, une barre de défilement peut être représentée en braille et la position du curseur peut être verbalisée : « le curseur est à 10% ». Une autre adaptation concerne l'ajout de liens de navigation supplémentaires, comme des tables de matières contenant tous les liens internes et externes d'un document. Il a été montré que ces liens sont très utiles pour faciliter la navigation pour ce groupe d'utilisateurs [Kennel 96] qui ne peut évidemment pas repérer rapidement les liens contenus dans le document.

- Prenons l'exemple d'un technicien de maintenance aéronautique équipé d'un terminal de type iPaq relié à un réseau sans fil. Ce technicien consulte la description d'une tâche de maintenance à travers son terminal. Cette description est présentée sous forme textuelle et accompagnée d'une vidéo montrant les différentes étapes de l'accomplissement de cette tâche. Puisque le terminal est équipé d'un écran de petite taille, cette vidéo est présentée de façon indépendante par rapport au reste des informations, ce qui n'est pas nécessairement le cas lorsque le document est consulté sur une station de travail. De plus, la taille de la vidéo doit certainement être réduite pour permettre son affichage dans l'écran.

- Pour les utilisateurs accédant à l'information à travers un portable connecté au réseau via un modem 56kbit/s, les informations de grand volume doivent être remplacées par des informations de plus petit volume tout en restant au maximum équivalentes. Par exemple, une vidéo d'une qualité proche du DVD, ayant une résolution de 640x212 pixels nécessite un débit moyen de 900kbit/s. La réduction à un débit de 56kbit/s peut se faire selon plusieurs méthodes. La première est d'agir sur la fidélité de la vidéo, c'est-à-dire changer la taille, la durée ou encore la qualité de la vidéo. La deuxième méthode est de changer de type de média, par exemple transformer la vidéo vers une suite d'images clés, ou encore vers un média de type audio en conservant uniquement la bande son de la vidéo [Mohan 99].

- Pour les utilisateurs accédant à l'information à travers un téléphone portable, si cette information n'est pas décrite dans le langage de marquage WML [WAF 99], alors elle

doit être convertie. Comme WML ne supporte qu'un nombre restreint de média, il est nécessaire de trouver un équivalent textuel ou imagé des informations véhiculées par les autres types de média.

Ces exemples permettent de faire ressortir un certain nombre de besoins en terme de *techniques d'adaptation*, comme le transcodage ou le filtrage de média. Ces techniques sont utilisées pour répondre au besoin ou à la contrainte induite par un paramètre d'adaptation. Ces exemples montrent aussi que des paramètres d'adaptation différents peuvent conduire au même résultat d'adaptation. Par exemple, pour une taille d'écran plus petite ou pour un faible débit réseau, une des adaptations est de réduire la taille des images afin qu'elles prennent moins de temps lors de la transmission et moins de place sur l'écran. Et à l'inverse, pour un même paramètre d'adaptation, plusieurs adaptations sont possibles. L'adaptation d'une vidéo pour permettre sa transmission sur un réseau à faible débit consiste, par exemple, au filtrage des images de la vidéo. Une autre solution est de la transformer en une suite d'images clés. Par la suite, en plus d'associer une technique à un paramètre d'adaptation, il convient donc de définir quelles sont les conditions de cette association.

1.5. Plan de la suite

Dans la suite de ce chapitre, nous décrivons les différents composants constituant le système adaptable pour répondre au problème de l'adaptation. Nous commençons par détailler le contexte utilisateur. Puis nous décrivons les infrastructures de base qui constituent un système d'informations multimédias. Enfin, dans la dernière section nous étudions quelques systèmes adaptables existants.

2. Modélisation du contexte de l'utilisateur

Comme nous l'avons vu dans la section 1.1, un des principaux composants du système adaptable est le contexte utilisateur. La modélisation de ce contexte permet de représenter les informations suivantes :

- les informations modélisant l'*utilisateur*, telles que son langage, ses capacités, ses préférences ou encore sa localisation physique ;
- les informations *logicielles et matérielles* représentant les caractéristiques du support de consultation du document (le terminal). Ces paramètres sont par exemple la taille de l'écran, la présence ou non de haut-parleurs, ou encore la vitesse du modem.

La connaissance précise de ces informations, que nous avons appelées paramètre d'adaptation, est essentielle afin de pouvoir les prendre en compte de façon optimale. Un des objectifs de cette section est de définir une taxonomie de ces paramètres. Grâce à cette taxonomie, nous associerons un ou plusieurs types d'adaptation en considérant une catégorie de paramètres (cf. chapitre 4) et non pas chaque paramètre indépendamment les uns des autres. Nous verrons en effet que le nombre de paramètres est relativement important.

Dans la suite de cette section, nous commençons par étudier les méthodes existantes pour modéliser un utilisateur. Puis dans la seconde sous-section, les modèles pour caractériser les terminaux sont décrits. Nous proposons ensuite notre propre taxonomie des différents paramètres d'adaptation. Enfin, dans la dernière section nous effectuons une synthèse de ces modèles.

2.1. Modèle utilisateur

La modélisation de l'utilisateur est étudiée dans de nombreux domaines de recherche. Ces domaines incluent principalement l'interface homme machine, l'intelligence artificielle, l'adaptation d'interface, les systèmes experts, et la présentation de document. La conséquence est l'existence d'une multitude d'approches et de techniques liées à la modélisation de l'utilisateur, et qui font l'objet de plusieurs états de l'art [Allen 90, Kobsa 93, Fischer 01]. Dans la suite de cette section, nous proposons une synthèse relatant les principaux concepts rencontrés dans ces domaines. Ces concepts sont classés en deux catégories : la modélisation proprement dite et l'acquisition de modèles utilisateur.

2.1.1. Modélisation

Modéliser l'utilisateur est une tâche ardue à cause du nombre très important des paramètres le caractérisant. Ces paramètres peuvent être regroupés selon plusieurs catégories, certaines de ces catégories impliquant des méthodes de modélisation particulières que nous décrivons. Nous donnons aussi pour chaque catégorie un petit exemple d'adaptation résultant de la prise en compte d'un ou de plusieurs paramètres. Nous proposons les catégories suivantes :

- **Carte d'identité**. Elle regroupe les informations concernant l'identité de l'utilisateur, par exemple son nom, son adresse, etc. Par exemple, lors de la consultation des films présents dans une salle de cinéma, ceux interdits aux moins de 12 ans ne seront pas présentés pour un utilisateur n'ayant pas au moins cet âge. Les paramètres constituant la carte d'identité peuvent être énumérés. Une telle énumération est présente en particulier dans le standard (prévu pour fin décembre 2001) P3P (Platform for Privacy Preferences [P3P 01]) défini par le W3C. Le principal objectif de ce standard est de définir un protocole pour informer l'utilisateur d'un site Web quelles informations personnelles sont utilisées par ce site. Pour cela, P3P définit en particulier un schéma de données XML représentant les données personnelles de l'utilisateur.

- **Localisation spatio-temporelle**. Il s'agit de modéliser l'*endroit* où se situe l'utilisateur au *moment* de l'accès à l'information. Par exemple, si l'utilisateur se situe en Europe, alors les prix seront affichés de préférence en Euros. S'il accède au site de l'office du tourisme d'une ville dans la soirée, alors la liste des restaurants les plus proches est affichée. S'il accède au même site en début d'après-midi, alors la liste des musées est présentée. Dans SWAN [Garlatti 99] est démontrée une utilisation de la localisation de l'utilisateur pour filtrer des informations dans un système d'informations maritimes.

- **Capacités cognitives et physiques**. La modélisation des capacités cognitives de l'utilisateur est utilisée notamment pour différencier les capacités d'apprentissage. La technique utilisée est la définition d'un ensemble de styles cognitifs. Les styles cognitifs sont des heuristiques de haut niveau qui organisent et contrôlent le comportement à travers une large variété de situations [Dufresnen 97]. À chaque utilisateur sont associés plusieurs styles cognitifs. Plus d'une vingtaine de styles cognitifs sont répertoriés dans la littérature [Ausburn 78], comme l'introversion et l'extroversion, l'impulsivité et la réflectivité, etc. Un autre aspect est la modélisation de la capacité physique des utilisateurs. Ces capacités peuvent elles aussi être énumérées. Des exemples d'adaptation induite par la prise en compte de cette catégorie de paramètres sont, entre autres, l'affichage d'informations concernant les accès à un bâtiment pour handicapés pour un

utilisateur dans une chaise roulante, l'emploi de couleurs avec un fort contraste pour un utilisateur daltonien, etc.

- **Émotion et humeur**. Plusieurs travaux s'intéressent à la modélisation des émotions et de l'humeur courante de l'utilisateur dans le cadre de l'adaptation [Ortony 88, Lisetti 00]. Par exemple, les capacités cognitives de l'utilisateur sont affectées selon qu'il est stressé ou non. Sept émotions de base sont recensées dans la littérature : la surprise, la peur, la colère, le dégoût, la tristesse, la joie et le mépris [Ekman 75].

- **Préférences**. Cette catégorie concerne la modélisation des intérêts de l'utilisateur vis-à-vis de la présentation et du domaine représenté par le document. Par exemple, un utilisateur consultant un site de recettes de cuisine peut préférer voir la réalisation de la recette étape par étape plutôt qu'une description globale. Lorsqu'il est en train de réaliser la recette dans la cuisine, il souhaite que le passage d'une étape à une autre soit guidé par la voix. Par contre, lorsqu'il est dans un lieu public et qu'il consulte une recette, il veut plutôt utiliser le sens tactile pour passer d'une étape à une autre. Pour le moment, il n'existe pas de modèle caractérisant les préférences liées à la présentation et qui soit indépendant du domaine.

- **Connaissances**. Elle concerne les connaissances générales de l'utilisateur. En fonction de ces connaissances, le système peut, par exemple, ajouter de l'aide supplémentaire pour un utilisateur débutant du système Unix [Chin 89]. Contrairement aux catégories précédentes, il est difficile d'énumérer les connaissances d'un utilisateur car elles dépendent fortement du domaine de l'application. C'est pourquoi des approches approximatives sur lesquelles il est possible de raisonner sont étudiées. Un exemple d'approche approximative, et probablement la plus populaire [Kobsa 93], est la définition de stéréotypes [Rich 79]. Un stéréotype contient quelques caractéristiques utilisateur pertinentes vis-à-vis de l'application. La définition de stéréotypes est effectuée par le concepteur de l'application et le nombre de stéréotypes dépend du domaine de l'application. Par exemple, la figure 22 représente une hiérarchie de stéréotypes dans le domaine médical. Chaque stéréotype contient des hypothèses sur la connaissance des termes techniques du domaine. D'autres approches existent pour modéliser les connaissances de l'utilisateur. Elles sont principalement étudiées dans le domaine de l'intelligence artificielle, notamment à travers l'activité de représentation de connaissances. Toutes ces approches reposent sur des représentations sur lesquelles il est possible d'utiliser différentes sortes de raisonnement (logique du premier ordre, raisonnement par incertitudes, résolution de conflit lorsque des assomptions contradictoires sont détectées).

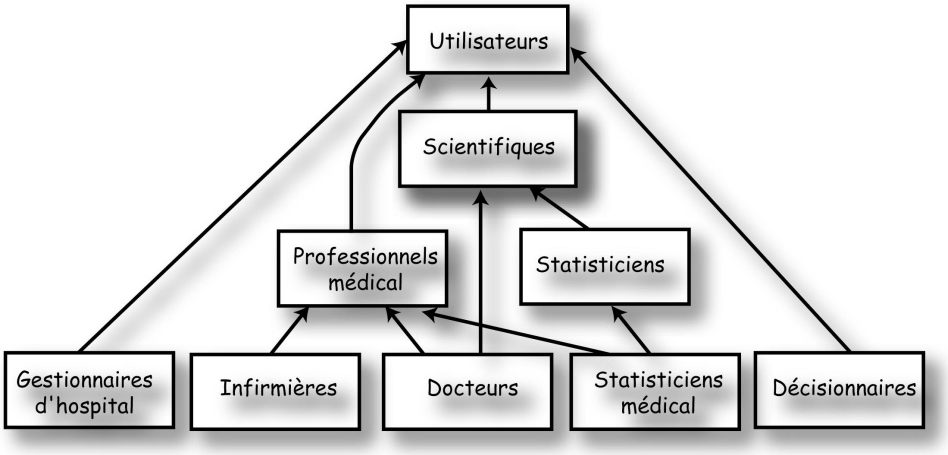


Figure 22. Exemple de stéréotype hiérarchique dans le domaine médical.

- **Objectifs et stratégie**. Cette catégorie rassemble les informations sur les objectifs de l'utilisateur et sur la façon dont il a l'intention de les accomplir. Avec ces informations, le système peut, par exemple, évaluer la stratégie actuelle de l'utilisateur et proposer de meilleures solutions. Un exemple de solution est l'ajout d'un lien de navigation vers la page cherchée. Les techniques associées à ce type de modélisation sont la reconnaissance d'objectifs [Carberry 88], la reconnaissance de stratégies [Kautz 87, Waern 96] et la génération de stratégies [Küpper 99].

Pour chacun des paramètres caractérisant l'utilisateur est associé un ensemble de valeurs. L'acquisition de ces valeurs peut se faire de plusieurs façons. Celles-ci sont étudiées dans la section suivante.

2.1.2. Acquisition

Un des problèmes difficiles de la modélisation de l'utilisateur est l'acquisition des caractéristiques de l'utilisateur. Deux types d'acquisition sont possibles [Chin 93] :

- **Explicite**. L'utilisateur interagit avec le système pour définir son modèle courant. Ce type d'acquisition est adapté pour connaître les préférences de l'utilisateur. Il existe de nombreuses applications sur le Web utilisant ce type d'acquisition (Amazon, Télérama, etc.) Cependant, lorsque le modèle comporte de nombreux attributs, cette méthode devient trop laborieuse pour l'utilisateur. Une approche automatique est alors nécessaire.

- **Implicite**. Le système tente de déduire automatiquement le modèle de l'utilisateur. En fonction des caractéristiques à déduire, plusieurs techniques pour construire le modèle ont été étudiées. Par exemple, des techniques d'observation du comportement de l'utilisateur, au niveau des liens qu'il utilise et du contenu qu'il regarde, sont utilisées pour déduire ses objectifs [Carberry 88]. Des techniques d'analyse d'expressions faciales sont étudiées pour déduire l'émotion courante ressentie par l'utilisateur [Lisetti 00]. L'identification d'un utilisateur vis-à-vis d'un stéréotype repose généralement sur un questionnaire initial puis sur l'usage de méthodes d'inférence [Rich 79].

L'acquisition du modèle de l'utilisateur fait partie d'un domaine de recherche à part entière. La description de l'ensemble des techniques relatives à ce domaine n'entre pas dans le cadre de

cette thèse. Cependant, les travaux qui visent à développer des *systèmes généraux* pour la modélisation de l'utilisateur [Kass 88, Kobsa 01a] (*General User Modeling Systems : GUMS)* présentent un intérêt particulier pour nos objectifs, puisqu'ils sont vus comme des composants réutilisables, en particulier, par un système adaptable. Ils gèrent alors l'acquisition du modèle de l'utilisateur de façon transparente pour le reste de l'application.

Le fonctionnement d'un GUMS est illustré figure 23. Pour chaque application, le GUMS garde une base de connaissance des propriétés des utilisateurs qui sont pertinentes pour l'application. L'application a la charge d'acquérir des faits nouveaux sur les utilisateurs et de les fournir au GUMS. Celui-ci vérifie la consistance de ce nouveau fait par rapport aux hypothèses (ou assomptions) maintenues dans le système, et informe l'application d'une éventuelle inconsistance. Il infère de nouvelles hypothèses. L'application a la possibilité d'interroger le GUMS pour obtenir des informations sur les hypothèses faites à propos de chaque utilisateur. Le GUMS peut aussi informer automatiquement l'application lorsque le modèle change.

Lors de la conception de l'application, le GUMS doit être rempli avec un modèle de connaissance de l'utilisateur relatif au domaine de l'application. Par exemple, pour une application médicale, le modèle représenté figure 22 peut être utilisé pour remplir le GUMS. La communication entre le GUMS et l'application s'effectue dans un langage propre au GUMS. Par exemple, dans le système BGP-MS [Kobsa 95], le message suivant

bgp-ms-tell (B S (B U (all x (-> (BALEINE x) (POISSON x))))))

signifie que l'utilisateur croit qu'une baleine est un poisson. Le caractère B est un opérateur signifiant une croyance (Belief). Les caractères S et U indiquent si la croyance est possédée par le système ou l'utilisateur.

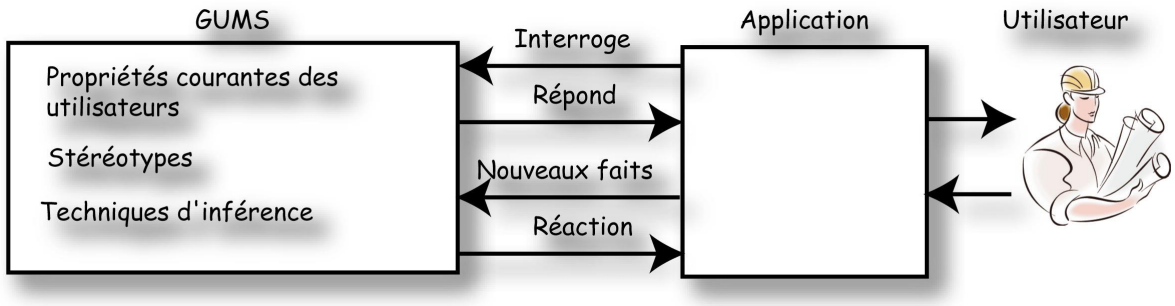


Figure 23. Architecture générale d'un système de modélisation de l'utilisateur.

La fonction principale d'un GUMS est de conserver une représentation des connaissances de l'utilisateur. Or, pour cette catégorie, ainsi que pour une partie de la catégorie préférences, les informations attachées à l'utilisateur sont dépendantes du domaine de l'application. Dans ce mémoire, nous n'exploitons pas ces modèles complexes pour deux raisons. La première raison est que la création et l'exploitation d'un modèle de connaissance est une tâche de très grande envergure. Pour s'en convaincre, le lecteur peut consulter, entre autres, le travail de définition et d'exploitation d'une ontologie pédagogique effectué par Sylvie Chabert-Ranwez [Chabert 00]. La seconde raison est que notre objectif est de proposer des solutions d'adaptation indépendantes du domaine représenté. Cependant nous intégrons le concept de GUMS dans notre système adaptable comme composant d'adaptation.

2.2. Modèles matériel et logiciel

Un système adaptable doit être en mesure de prendre en compte les caractéristiques du terminal qui sert à la consultation d'un document multimédia. Pour cela, il doit connaître ses caractéristiques matérielles, comme le type de processeur, la taille de l'écran ou encore le débit théorique d'un modem. Il doit aussi connaître l'environnement logiciel du terminal, tel que le système d'exploitation, les types de formats de média supportés (JPEG, GIF, etc.), ou encore la version des langages reconnus (XHTML, HTML 4.0, HTML 3.2, etc.).

Plusieurs institutions ont défini un ou plusieurs vocabulaires pour représenter ces informations et une structure pour les organiser [FIPA 01, RFC2506 01, MQ 01, UAProf 99]. Dans le contexte du Web, l'uniformisation et la standardisation de ces vocabulaires et de ces structures sont des tâches critiques pour garantir l'interopérabilité des multiples systèmes qui ont besoin de ces vocabulaires. Au moment de l'écriture de ce mémoire, cette standardisation est en cours de définition au sein du W3C à travers la (future) recommandation appelée CC/PP (Composite Capabilities/Preference Profiles [CC/PP 01]). Comme l'objectif de ce mémoire n'est pas de décrire l'ensemble des spécifications, nous nous focalisons sur CC/PP puis nous proposons une classification des attributs qui vont probablement le constituer.

De façon générale, CC/PP définit une *structure* et un *vocabulaire* pour caractériser les capacités du terminal, appelé *profil*. CC/PP permet aussi de décrire le comportement de *proxy*, c'est-à-dire un ensemble de services intermédiaires entre le client et le serveur. La syntaxe de CC/PP repose sur RDF (cf. section 2.2.4 du chapitre 2).

La structure d'un profil CC/PP est composée d'un certain nombre de *composants* représentés par des nœuds RDF. Chaque composant est caractérisé par des attributs CC/PP. Par exemple, la figure 24 contient deux composants : le composant *hardware* et le composant *software*. Les attributs de *hardware*, représentés par des arcs étiquetés, sont *memory*, *bluetooth* et un ensemble de valeurs par défaut. La valeur des attributs est typée et respecte la syntaxe définie dans [SchemaDatatype 01].

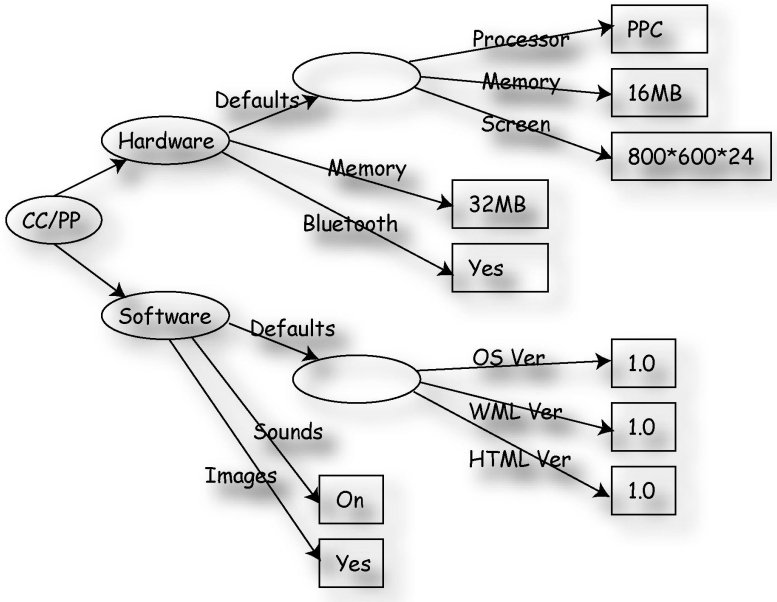


Figure 24. Exemple de modélisation en utilisant CC/PP.

Les attributs définis par les vocabulaires candidats dans ce standard peuvent être groupés selon qu'ils caractérisent les matériels ou les logiciels. Les attributs du premier groupe sont par exemple le type de processeur, la vitesse du modem, le nombre et la position des enceintes, le support du décodage DTS par la carte son et la résolution de l'écran. Nous avons vu dans la section 1.4 de ce chapitre quelques exemples d'adaptation en fonction de ces paramètres.

Le second groupe inclut les paramètres caractérisant l'environnement logiciel du terminal. Ces caractéristiques regroupent les attributs définissant les applications. Par exemple, le forum Wap propose un schéma RDF pour décrire les caractéristiques d'un navigateur. Les attributs de ce schéma sont, entre autres, le nom du navigateur, les formats de fichier qu'il accepte et de façon plus fine la version HTML supportée. En plus de caractériser les applications, les fonctionnalités des documents sont aussi décrites. En XML, ces fonctionnalités sont associées à un espace de noms. Par exemple, l'espace de nom http://www.w3.org/TR/SMIL #animation correspond à la fonctionnalité d'animation du langage SMIL 2.0. L'intérêt est de pouvoir déterminer de façon fine les fonctionnalités supportées par un terminal pour permettre une adaptation plus fine. C'est pourquoi il y a une tendance pour modulariser les langages existants. Les exemples les plus connus sont la modularisation de XHTML et de SMIL.

2.3. Taxonomie

À travers cette section, nous avons montré que l'adaptation doit se faire pour un nombre important de paramètres. Les considérer un par un lors de la conception d'un système adaptable est une tâche titanesque. C'est pourquoi il est indispensable de les classer pour ainsi associer un traitement d'adaptation à une catégorie de paramètres d'adaptation.

Une classification possible est de distinguer la fréquence de changement de la valeur des paramètres d'adaptation. Nous verrons par la suite que cette classification est utile pour généraliser ces systèmes à l'ensemble des paramètres d'adaptation. Trois catégories sont identifiées :

- Statique : la valeur du paramètre est constante tout au long de la session de consultation d'un document (carte d'identité de l'utilisateur, capacités cognitives et physiques, etc.) ;
- Dynamique : la valeur d'un paramètre est amenée à changer durant la consultation d'un document. Ce changement intervient de façon peu fréquente (changement de la taille de la fenêtre, préférence entre une présentation interactive ou non, etc.) ;
- Temps réel : la valeur d'un paramètre change constamment durant la présentation du document. C'est le cas, par exemple, du débit réseau et de la charge CPU.

Un autre critère de classification discriminant est le type du paramètre. Par exemple, un paramètre de type énuméré comme les styles cognitifs n'est pas traité de la même façon qu'un paramètre de type continu (entier, réel, etc.). Des techniques de cas sont utilisées pour le premier type, tandis que des techniques à base de calcul sont utilisées dans le second cas.

Enfin on peut noter qu'il y a deux catégories de paramètres d'adaptation : ceux liés au domaine et ceux liés à la présentation. Dans le premier cas, la prise en compte du paramètre d'adaptation dépend du domaine. Par exemple, la présentation d'un exemple avant le théorème dépend de ces deux notions. Dans le deuxième cas, la prise en compte d'un

paramètre d'adaptation ne dépend que de la présentation, comme l'utilisation de couleur à fort contraste pour les daltoniens.

2.4. Conclusion

Dans cette section nous avons décrit un certain nombre de paramètres caractérisant le contexte de l'utilisateur. Nous avons ensuite proposé une taxonomie pour simplifier la tâche de conception d'un système adaptable. L'intérêt de cette taxonomie ne se limite pas uniquement à cet aspect. En effet, l'ensemble des paramètres composant le contexte utilisateur n'est pas un ensemble figé. Les matériels évoluent, de nouveaux périphériques apparaissent, les modèles utilisateur s'affinent, etc. Les modèles, surtout ceux caractérisant le matériel et les logiciels doivent être à la fois flexibles et extensibles. C'est le cas notamment de CC/PP. Au niveau de la conception du système adaptable, celui-ci doit être suffisamment générique pour permettre de prendre en compte ces nouveaux paramètres, dans le meilleur des cas, sans remettre en cause de façon fondamentale le système d'information. L'usage d'une taxonomie peut aider à la conception d'un tel système.

Comme nous pouvons le constater, les modèles utilisateur, matériel et logiciel utilisent des formats de représentation différents. Généralement, les modèles utilisateur reposent sur un format ad hoc, hormis P3P qui modélise les données utilisateur en XML. Du côté de la modélisation du matériel et des logiciels, l'usage de RDF à travers CC/PP semble acquis. À terme, l'unification de ces modèles est souhaitable afin de canaliser l'effort de conception sur un seul et même modèle.

Lors de la conception d'un document adaptable, seuls les paramètres dépendants de du domaine nécessitent une phase d'identification pour les prendre en compte : les paramètres de présentations sont indépendants du domaine et peuvent donc déjà être identifiés.

Maintenant que nous avons identifié les paramètres constituant le contexte de l'utilisateur, nous décrivons les techniques existantes pour les prendre en compte au sein du système adaptable.

3. Infrastructures de base pour les systèmes adaptables

Dans cette section, nous présentons un ensemble de moyens permettant de dériver plusieurs présentations. Ces moyens sont classées en trois catégories selon leur niveau fonctionnel (cf. figure 25) : les techniques d'adaptation au niveau du système et réseau, au niveau du modèle de données et au niveau de l'interface utilisateur. Pour chacun des moyens présentés dans cette section, nous nous attachons à faire ressortir les points suivants :

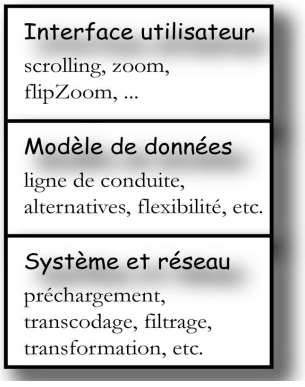


Figure 25. Infrastructures d'adaptation en trois couches.

- paramètres d'adaptation : certains des moyens présentés sont spécifiques à un ensemble de paramètres. Dans ce cas, nous les énumérons explicitement ;
- dynamique versus statique : mesure la capacité à prendre en compte un paramètre d'adaptation

pendant l'exécution du document (dynamique) ou au contraire avant l'exécution du document (statique) ;

- influence sur l'organisation des médias : la modification de l'organisation des médias représente un changement important qui peut conduire vers une présentation inexploitable (cf. section 1.3).

3.1. Techniques d'adaptation

Dans cette section nous décrivons les techniques utilisées par le système pour adapter la présentation d'un document. Ces techniques correspondent au premier niveau identifié figure 25.

3.1.1. Système et réseau : qualité de service

De façon générale, la qualité de service vise à améliorer le degré de satisfaction de l'utilisateur d'un service. Dans notre contexte, la qualité de service apporte des solutions aux problèmes posés lors de la consultation de document à travers des réseaux et des systèmes hétérogènes.

L'usage du réseau induit, en particulier, des problèmes de latence et de temps de chargement des média. La technique de pré-chargement [Ng 96, Laforge 01] repose sur un constat simple : comme le temps de récupération des objets médias à travers un réseau est non négligeable, il est nécessaire de récupérer ces données antérieurement à leur utilisation effective.

Cette technique est notamment exploitée dans les présentations SMIL 2.0 à travers l'élément prefetch. L'auteur a la possibilité de demander de pré-charger une partie d'un média lors de la présentation. Cette partie correspond soit à une durée précise du média (attribut mediaTime), ou soit à une taille précise (attribut mediaSize). De plus, l'auteur peut préciser la bande passante maximum à utiliser (attribut bandwidth). Grâce à cet élément, l'auteur peut prévoir de pré-charger un média tout en affichant un message du type : « Chargement en cours… ». L'ajout de l'élément prefetch peut aussi se faire de façon automatique grâce à des outils d'analyse de la bande passante requise par les différents médias du document. Néanmoins, l'inconvénient de cette approche, qui inclut des primitives de pré-chargement dans le document, est qu'elle fait l'hypothèse que les ressources sont fixes et n'évoluent pas dans le temps. Or dans la réalité, les ressources varient d'une part en fonction du temps et d'autre part en fonction du contexte utilisateur. Une approche plus dynamique est donc nécessaire pour garantir une meilleure qualité de service et répondre au problème d'adaptation. Le document est analysé au moment de la présentation et le pré-chargement s'ajuste alors en fonction des ressources disponibles [Laforge 01].

Au niveau système, le document doit s'adapter aux ressources système disponibles, à la bande passante des entrées/sorties avec les disques durs, aux cycles CPU, à l'espace mémoire disponible, etc. Chaque système est différent et pour garantir une exécution uniforme sur l'ensemble de ces systèmes, des mécanismes d'allocation de ressources sont mis en place. Cette allocation peut être statique ou dynamique pour s'adapter aux besoins courants. De nombreux travaux portent sur ce problème d'allocation de ressources. Le lecteur peut trouver plus amples informations dans [Zhang 93].

Ces techniques agissent sur la gestion de ressources système et réseau, mais n'ont pas d'influence sur l'organisation des médias dans le document. Elles permettent d'améliorer la qualité de la présentation sur des systèmes et des réseaux hétérogènes.

3.1.2. Codage multiniveau pour les médias

Le principe d'une représentation multiniveau est de permettre la génération de plusieurs flux (de type image ou vidéo) de débit et qualité différents, ceci à partir d'une source donnée [Dabbous 01]. Les informations de codage sont organisées en plusieurs couches : la couche de base et plusieurs couches de raffinement (cf. figure 26). Trois découpages sont possibles :

- Découpage temporel : permet de produire des flux de différentes qualités. Par exemple, dans les formats vidéo de type IBP, les images B sont des images de raffinement. C'est ce type de découpage qui est montré dans la figure 26.
- Découpage spatial : permet de produire des flux de différentes résolutions. Chaque nouvelle couche raffine la résolution de la couche de base.
- Découpage SNR : définit une hiérarchie de qualité en terme de rapport signal à bruit et donc une hiérarchie de débits associés. Les couches de raffinement correspondent aux corrections d'erreur des couches inférieures.

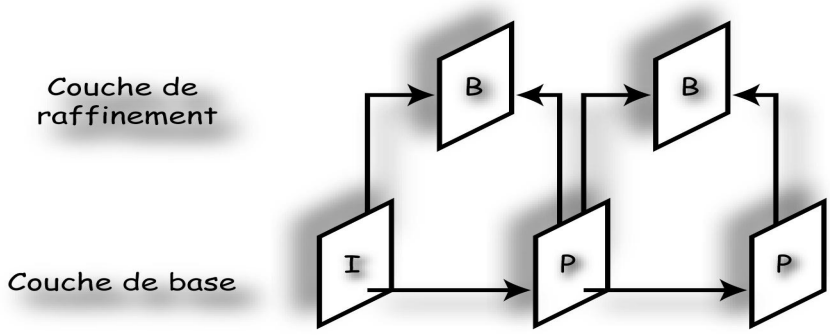


Figure 26. Multi-niveau temporel dans le norme H.263 version 2 et MPEG-4 vidéo.

Ces découpages peuvent être combinés afin de générer des flux de qualité et de résolution variées. On peut noter que plusieurs formats de média actuels reposent sur un codage multiniveau : MPEG-4 Vidéo [ISO 98], H.263 version 2 [ITU 98], JPEG-2000 [Xiong 00], etc.

Le codage multiniveau ne remet pas en cause l'organisation des médias. Le découpage spatial permet de produire des images de résolution différente mais l'utilisation actuelle de ce type de codage vise à obtenir une image restituée de façon progressive. Pour le changement de taille, la technique de transcodage (cf. section suivante) est préférée car elle permet un changement de taille à une granularité plus fine. Le codage multiniveau est cependant très intéressant pour réduire le débit nécessaire à la transmission d'un flux de qualité dégradée, et ceci sans surcoût CPU de la part du serveur.

3.1.3. Transcodage de média

De façon générale, le principe du transcodage est de convertir un ensemble d'informations d'une représentation vers une autre. Dans cette section nous nous focalisons plus

particulièrement sur le transcodage appliqué aux médias. Ce type de transcodage inclut les trois types de conversion suivants :

- la conversion du format de représentation d'un média (par exemple, une image GIF vers une image WBMP). Ce type de conversion s'effectue en conservant au maximum les caractéristiques du média source lorsque le format cible le permet. Cependant, dans certains cas, ce type de conversion entraîne des pertes d'informations, comme la couleur lors de la conversion d'une image GIF vers une image WBMP de type 0 [WAF 99]. Si la couleur a une valeur sémantique, alors il est important de la représenter par exemple en utilisant un descriptif textuel ;

- la conversion d'un média vers un autre type de média (un texte vers un son). Le tableau 2 donne un aperçu des types de conversion pour les principaux types de média : le texte, l'image, la vidéo et le son. Par exemple, pour diminuer la bande passante requise par le flux d'une vidéo, celle-ci peut est transformée en une suite d'images ou, à l'extrême, en un texte équivalent. Généralement, les principaux types de transcodage étudiés dans la littérature et utilisés dans les outils visent à réduire soit l'espace nécessaire pour présenter le média, soit la bande passante du réseau. Cependant, d'autres types de transcodage sont nécessaires pour traiter le problème d'adaptation dans sa globalité, comme la transformation d'un texte avec du style vers une image. Ce type de transcodage est utilisé lorsque le langage de présentation cible ne peut pas représenter du texte avec du style mais par contre peut représenter des images. C'est le cas, entre autres, du langage SMIL 2.0 (cf. conversion xhtml+smil vers SMIL 2.0 dans la section 5.1 du chapitre 4) ;

- la distillation, c'est-à-dire le fait d'agir sur la compression des médias en perdant de l'information [Fox 96, Yeadon 96]. Ces pertes sont, par exemple, la réduction/augmentation de la taille spatiale du média, la réduction du nombre de couleurs, le passage d'un son stéréo vers un son mono, ou encore la réduction d'un texte en conservant uniquement les mots clés [Björk 99]. À la différence du codage multiniveau, la qualité du média peut être contrôlée de façon plus fine, en non sous forme de palier (couche). Cependant ce contrôle a un coût au niveau du temps de traitement de la distillation.

Le transcodage permet de présenter le contenu d'un média sous différentes formes et ainsi, permettre aux terminaux n'ayant pas la capacité de présenter le contenu sous la forme initiale de le faire (format non supporté, enceintes non présentes, etc.) sous une forme alternative. Il permet aussi, dans certains cas, de réduire la bande passante et la charge CPU requises pour jouer un média. Au niveau de l'utilisateur, le transcodage peut être utile pour répondre à ses préférences et aussi à ses déficiences (synthétiser de la parole plutôt que d'afficher un texte).

Type de média source	Type de média cible	Technique(s) utilisée(s) pour le transcodage
Texte avec du style	Texte	Réduction de texte [Bickmore 99], Stretchtext [Boyle 94]
	Audio	Synthèse vocale
	Image	Formatage + génération image bitmap
Image	Texte	Analyse du contenu de l'image [Smith 98]
Audio	Texte	Reconnaissance vocale [Rabiner 93]
Vidéo	Audio	Filtrage des informations visuelles
	Image	Analyse d'images clés [OC 91]
	Texte	Analyse du contenu des images [Smith 98]

Tableau 2. Exemples de transcodage inter-média.

Le transcodage permet d'adapter la présentation de façon dynamique : pendant l'exécution du document multimédia, des techniques de transcodage différentes peuvent être appliquées pour un même média. Cela permet de s'adapter aux variations de ressources pendant l'exécution de la présentation. Par exemple, si au cours de la présentation le débit réseau diminue de telle façon critique, alors celle-ci peut-être transformée en une suite d'images.

Certains types de transcodages remettent en cause à la fois l'organisation temporelle et spatiale du document. Par exemple, la conversion d'un texte vers un son nécessite à la fois de modifier la durée d'exécution du média, d'ajouter un média de type son au document et de libérer l'espace graphique occupé précédemment par le texte. La modification de la durée ne peut se faire que sous certaines conditions : la spécification de l'auteur doit être respectée ou, dans le cas de l'adaptation par transgression, le scénario doit être cohérent. L'ajout de média, comme un son, nécessite de déterminer, soit par l'auteur, soit automatiquement, un emplacement dans le scénario qui n'entre pas en conflit avec les médias existants. À notre connaissance, aucun travail portant sur le transcodage n'adresse le problème de la réorganisation temporelle et/ou spatiale de façon globale. Cependant, nous verrons dans la section suivante quelques éléments de réponse grâce aux méthodes de spécification flexible (cf. section 3.2.3), d'extension (cf. section 3.1.5) et de filtrage.

3.1.4. Filtrage

Le filtrage est un cas particulier du transcodage auquel est ajouté le média « nul » pour supprimer le média source de la présentation. Cependant, à la différence du transcodage, qui cherche à conserver le contenu véhiculé par le média, aucune information n'est conservée par le filtrage. Il convient donc d'avoir des moyens pour évaluer la pertinence du média pour justifier ou non son filtrage.

Dans [Paek 98] et [Smith 98], le filtrage d'images repose sur la détection de l'objectif de l'image. Le contenu de l'image est analysé puis l'image est classée selon différents critères comme : une image publicitaire, une image avec du contenu textuel, une image décorative (bouton, règle, etc.), une image informative (en construction, avertissement, etc.), un logo, ou encore une image pour naviguer (retour, avance, etc). Dans les systèmes tels que [Freire 01] et myhahoo[14], ou de façon générale les sites qui proposent de personnaliser la présentation et le contenu, le filtrage est effectué manuellement par l'utilisateur, en sélectionnant les parties du document qu'il juge sans intérêt.

En ce qui concerne l'influence sur l'organisation des médias, les systèmes actuels reposent sur des gestionnaires de mise en page génériques. La suppression n'a donc pas de conséquence sur la cohérence de la présentation. Cependant, pour les documents multimédias, en particulier, qui permettent de positionner les médias relativement aux autres médias, le filtrage ne peut pas se faire aussi simplement. Les relations qui lui sont associées sont elles aussi supprimées ce qui peut engendrer des incohérences visuelles. Par exemple, supposons que sur la figure 21 l'auteur a positionné les trois médias, que nous appelons A B et C, décrivant une phase de la lune de façon relative : le média le plus à gauche A est positionné de façon absolue et les relations entre A et B et entre B et C sont spécifiées. Si le média du milieu B est supprimé, les deux relations sont supprimées. Le média C n'ayant pas de positionnement absolue est alors montré à son emplacement par défaut. Par contre si le filtrage s'effectue en calculant les relations induites, alors le média C sera bien positionné.

3.1.5. Extension

L'extension consiste à ajouter dans le document un média ou un fragment de document. Cette technique est utilisée essentiellement dans les systèmes d'aide à la navigation [Anderson 01, Rossi 01]. Ces systèmes sont tous basés sur HTML et à chaque fois c'est le concepteur du site qui décide où positionner les liens générés automatiquement.

3.1.6. Transformation

D'un point de vue syntaxique, la transformation permet de réorganiser les éléments du document en les filtrant, en les déplaçant, en les renommant ou en générant de nouveaux éléments. D'un point de vue de la présentation, la transformation permet de réorganiser complètement la composition des médias. Enfin, d'un point de vue langagier, la transformation permet de convertir un document écrit dans un format donné vers un autre document écrit dans un autre format (par exemple HTML vers WML).

Nous avons vu dans le chapitre 2 que la transformation repose sur une spécification explicite des règles de transformation. Ces règles offrent la liberté à un auteur de spécifier les

[14] my.yahoo.com

transformations qu'il souhaite et par conséquent offrent une grande latitude de spécification. Cette liberté à un prix qui se mesure en temps et en niveau de difficulté pour spécifier une transformation. Cependant, il est possible d'identifier plusieurs transformations récurrentes. Nous étudions ces transformations à travers plusieurs exemples d'utilisation.

Dans [Bickmore 99] la transformation est utilisée pour donner un aperçu de la présentation : le document original est allégé puis découpé selon des règles précises. Pour alléger le document, Bickmore utilise la technique *d'élision syntaxique* de média. Par exemple le texte d'un paragraphe est transformé en un texte ne contenant que son début et en un lien pointant vers le texte original. Dans le cas d'une image, la taille de celle-ci est réduite puis un lien est ajouté vers l'image originale. Bien que Bickmore l'utilise que pour les médias texte et image, ce type de transformation peut être généralisé à l'ensemble des médias visibles.

Bickmore étend la technique d'élision syntaxique aux sections HTML H1, H2, etc. Le contenu de chaque section est supprimé et l'en-tête de section est converti en un lien hypertexte. Lorsque le document comporte plusieurs niveaux de section, la conversion peut s'effectuer à différents niveaux de granularité. Selon le niveau utilisé, l'espace graphique requis est plus ou moins important. Par exemple, la figure ci-dessous montre le découpage d'une page contenant trois sections.

Dans [Anderson 01] sont proposées trois transformations type : « elide-content », « swap-siblings » et « add-shortcut ». La première transformation remplace un sous-arbre par un lien vers le contenu original. Elle est utilisée lorsque le contenu est rarement accédé par l'utilisateur. La deuxième transformation permute deux sous-arbres qui sont frères. Cela permet d'afficher en priorité des informations jugées intéressantes pour le lecteur. Enfin la troisième transformation ajoute un nouveau lien dans la page Web. Cet opérateur est utilisé pour avoir un accès direct à un document qui, à l'origine, est accessible en passant par plusieurs autres documents. Ces transformations permettent de faciliter la navigation du lecteur.

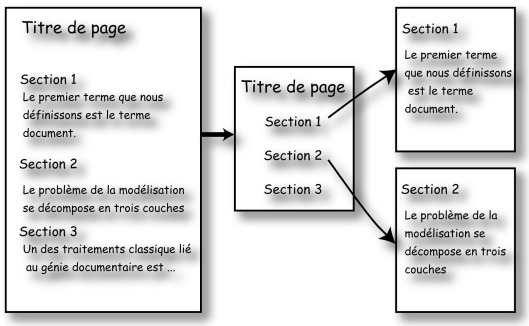


Figure 27. Exemple de transformation basée sur l'organisation en sections.

Des expériences de transformations à plus grande échelle ont été menées principalement pour la conversion de document HTML vers WML [Kaasinen 00, Chen 01]. Dans [Kaasinen 00] la conversion se fait directement à partir du document HTML. La transformation se fait selon des règles précises, qui sont les suivantes :

- pour les tables, trois méthodes de conversion sont utilisées selon la complexité de la table à convertir :

- pour de petites tables, la conversion conserve la table originale ;
- pour des tables imbriquées, puisque WML ne supporte pas cette fonctionnalité, le convertisseur les transforme en liens. En cliquant sur le lien, le contenu de la cellule est affiché dans une carte (card) séparée ;
- enfin la dernière méthode consiste à convertir la table en liste [Amaya].

- les images sont représentées par un texte en utilisant soit l'attribut `alt`, soit le nom du fichier image ;
- les cadres (`frame`) HTML sont convertis en un ou plusieurs jeux (`deck`) de cartes WML.

Ce type de conversion fonctionne relativement bien à condition que les pages HTML respectent un ensemble de règles de conception [Kaasinen 00]. En particulier, les tables sont souvent utilisées pour mettre en page le document. Un autre problème est que le découpage en jeux de carte repose uniquement sur les informations de présentation. Or ces informations sont souvent en trop grande quantité pour être contenues sur un écran plus petit que celui prévu.

Dans tous les cas, la transformation s'appuie sur une certaine connaissance. Cette connaissance peut être soit contenu dans le document source lui-même, comme les titres de section. Elle peut être aussi induite par l'observation de l'usage du document par l'utilisateur. La transformation repose donc sur la modélisation de ces données ou méta-données.

On peut noter que la transformation peut s'appliquer à tous les types de connaissances. Par conséquent, c'est certainement la technique la plus apte pour prendre en compte la plupart des paramètres d'adaptation autres que ceux liés à la présentation.

3.2. Spécification pour l'adaptation

Le champ d'application des techniques décrites précédemment est limité soit à cause du manque d'expressivité du document à adapter, soit à cause de la faiblesse des méthodes d'analyse. Dans cette section nous présentons les techniques qu'un auteur peut utiliser pour augmenter le pouvoir d'action des techniques précédentes. Nous présentons aussi quels sont les outils à la disposition de l'auteur pour spécifier un document de présentation adaptable, indépendamment des techniques précédentes. L'implantation de certains de ces outils nécessite l'usage de techniques que nous identifions.

3.2.1. Ligne de conduite

Une ligne de conduite est un ensemble de règles informelles à suivre, en particulier pour construire un document. Le W3C a édité plusieurs lignes de conduite pour produire des documents accessibles par des personnes avec des déficiences [WebAccess 99]. Ces règles s'adressent à la fois aux éditeurs de documents, ainsi qu'aux concepteurs d'outils d'édition. Ainsi, il existe une règle qui est de fournir un texte équivalent à tous les autres contenus non textuels. Une autre règle est de s'assurer qu'un texte ou une image soient compréhensibles sans couleur.

Certaines lignes de conduites du W3C peuvent être formalisées, comme la nécessité de fournir une alternative textuelle. Cependant, la majorité des lignes de conduite du W3C porte sur le

contenu même du document, comme s'assurer que le document est simple et clair. Pour ces règles, il est difficile, voir impossible, de mettre en place des moyens automatiques pour contrôler le respect ou non des lignes de conduite.

Dans la section 3.1.5, nous avons vu que les tables peuvent être correctement converties en listes si elles sont « bien formées ». Les images sont converties en texte à condition que l'attribut `alt` soit spécifié. Les lignes de conduites peuvent dans ce cas être utilisée pour permettre l'application de la transformation de table en liste. Cependant, pour être réellement utilisable, elle doit aussi être formalisable.

3.2.2. Alternatives

Le principe de l'alternative est de proposer une solution de remplacement pour un média, pour une partie du document, ou pour le document complet. Par exemple, dans HTML la spécification d'une alternative textuelle s'effectue en utilisant l'attribut `alt`. Dans le langage SMIL 2.0, les alternatives sont spécifiées par l'élément `switch`. Chacune des alternatives est représentée par une paire attribut-valeur associée à un choix de présentation correspondant à un fragment de document.

Pour ces deux langages, l'alternative reste statique : elle est spécifiée au moment de l'édition du document. C'est pourquoi, le modèle ZYX [Boll 00] propose une extension à SMIL2 en permettant une adaptation dynamique à base d'alternatives. Cette extension repose sur l'ajout de l'élément `query` permettant de faire une requête qui sera évaluée durant la présentation du document.

De façon générale, la spécification d'alternatives s'effectue manuellement par un auteur. L'usage de techniques de transcodage (cf. section 3.1.3) permet de calculer automatiquement des alternatives, et ainsi augmenter [Boll 00] le document original. L'usage de cette technique accroît l'intérêt de l'alternative puisqu'elles ont tendance à être générées automatiquement ou semi-automatiquement : la charge de l'auteur est alors nettement diminuée. On peut noter cependant que le calcul automatique d'alternatives se fait dans la limite des capacités des techniques de transcodage.

3.2.3. Feuille de style

Les feuilles de style sont utilisées pour fournir plusieurs styles de présentation à un même document. Par exemple, les feuilles de style définissant des couleurs à fort contraste seront employées par les daltoniens. Pour le moment, l'usage des feuilles de styles se limite à la proposition de plusieurs styles de présentation. Aucune utilisation n'est recensée dans le contexte de l'adaptation.

3.2.4. Flexibilité

Pour pouvoir changer la durée d'un média tout en respectant la spécification de l'auteur, une solution est d'ajouter de la flexibilité. La spécification de la flexibilité peut se faire à deux niveaux : au niveau du média, et au niveau de la composition des média.

Madeus (cf. section 2.2.3.5 du chapitre 2) permet de spécifier des présentations multimédia flexibles à la fois sur les médias et sur la composition des médias. La durée de chaque média est spécifiée par un triplet `[durMin, durPref, durMax]` représentant un intervalle de durée. Au

niveau de la composition, Madeus permet de spécifier des relations temporelles et spatiales sans contraindre obligatoirement la durée ou l'espacement des deux objets mis en relation. Par exemple, la relation `A before B` signifie simplement que A doit se jouer avant B : aucune précision n'est donnée sur *quand* A doit se jouer avant B.

Le langage SMIL 2.0 permet aussi de spécifier une borne minimum et une borne maximum représentant un intervalle de durée.

Dans [Badros 01] est proposé une extension à SVG pour spécifier des contraintes. Par exemple, la spécification suivante :

```
1. <constraint rule="rect_w >= rect_h" strength="strong"/>
2. <rect x="10" y="20" width="rect_w" height="rect_h"/>
```

signifie que le rectangle doit être au moins plus large que haut. Lors du formatage, plusieurs solutions existent, chacune respectant la spécification de l'auteur. Plusieurs autres langages ont été étendus par l'ajout de contraintes : CCSS [Badros 99] et VRML [Diehl 00].

On peut remarquer que la flexibilité au niveau de la composition est monodimensionnelle. Aucun travail à notre connaissance ne décrit une flexibilité multidimensionnelle (cf. section 4.2 du chapitre 4).

3.2.5. Méta-données

Certaines des techniques d'adaptation que nous avons étudié dans la section 3.1 repose sur la spécification de méta-données. Par exemple, certain type de filtrage repose sur la connaissance de la catégorie d'une image : logo, décoration, schéma, etc. Ces méta-données sont soit spécifiées par un auteur, soit générées par un outil ou soit une combinaison des deux.

Dans [Vetro 01] le système présenté utilise les méta-données associées à une vidéo pour ne transmettre que les objets intéressants. La figure 28 montre l'adaptation d'une vidéo à laquelle l'image en arrière plan, jugée de moindre importance, a été filtrée. La conséquence est le la réduction de la bande passante nécessaire pour transmettre la vidéo. Ce système repose sur les schémas de description de vidéo définis dans MPEG7 [ISO 01].

Figure 28. Illustration d'une vidéo adaptée en utilisant une identification des objets clés.

On peut noter l'initiative de la part de Dublin Core[15] est de standardiser plusieurs méta-données relatives à de nombreux domaines comme l'éducation, les manuels utilisateur et les bibliothèques. Un des intérêts de cette initiative est de permettre le développement de techniques d'adaptation réutilisables sur l'ensemble des documents caractérisés par ces méta-données.

3.2.6. Composants graphiques multimodaux

Une autre technique est de spécifier dans le document directement des composants graphiques adaptables. Ces composants multimodaux offrent une modalité de sortie directement adaptée, en particulier, aux petites tailles d'écran tout en préservant l'utilisabilité. C'est l'approche exploitée par J. Coutaz, G. Calvary et D. Thevenin et [Thevenin 99, Calvary 01]. La figure 29 donne un exemple d'un composant multimodal pour spécifier un jour. La première modalité se caractérise notamment par un coût visuel plus important par rapport à la seconde modalité

Les composants multimodaux s'apparentent à la technique transformation qui, à partir de la donnée jour, peut produire plusieurs présentations. Cependant, les composants multimodaux ont l'avantage d'être très performants lorsqu'il s'agit de passer d'une modalité à une autre. Cela est très utile par exemple lorsque l'utilisateur change la taille de sa fenêtre. L'usage de la transformation est, dans ce cas, plus difficile à mettre en œuvre car celle-ci nécessite généralement un temps d'exécution relativement long, non adapté à un usage interactif. On peut noter cependant que les composants multimodaux sont prédéfinis et ne répondent donc pas forcement à l'ensemble des besoins de l'auteur. De plus, ils ne prennent pas en compte la dimension temporelle pour réaliser l'adaptation.

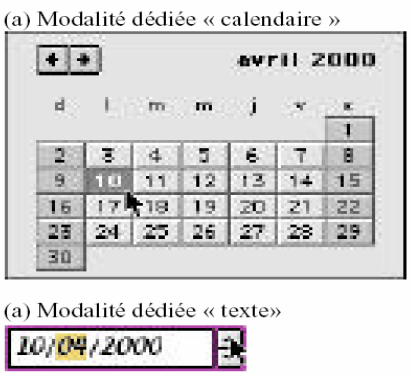


Figure 29. Deux modalités dédiées à la spécification d'un jour.

3.3. Interaction de l'utilisateur pour l'adaptation

Lorsque l'adaptation ne peut pas se faire ni au niveau du système, ni au niveau de l'auteur, le dernier recours est de la faire au niveau de l'utilisateur. L'exemple classique est l'utilisation de barre de défilement pour visualiser les parties du document non visibles. Un autre exemple est l'utilisation de loupe pour agrandir ou rétrécir l'affichage. Dans [Hsu 94] est relatée l'expérimentation d'une table des matières active dans un navigateur pour permettre à

[15] Dublin Core Metadata Initiative, http://dublincore.org/.

l'utilisateur de développer ou de réduire le niveau de détails dans un document. D'autres techniques entrent dans cette catégorie, comme l'usage de composants graphiques semi transparents [Kamba 96] pour optimiser l'espace graphique. L'utilisateur peut décider de rendre une barre d'outils transparente et ainsi allouer de l'espace au contenu du document. La technique appelée flipZoom [Björk 99] permet de montrer plusieurs images sous la forme de vignettes et de zoomer sur l'une d'entre elles dans la même fenêtre : la taille des vignettes est réduite et elles sont réorganisées pour permettre l'affichage de l'image sélectionnée.

3.4. Conclusion

Dans cette section, nous avons présenté plusieurs infrastructures pour l'adaptation de document. Ces infrastructures agissent soit au sein du système d'informations, soit au niveau du contexte de l'utilisateur par le terminal ou par l'utilisateur. Nous avons étudié ces infrastructures de façon indépendante. Cependant, la réalisation d'un système adaptable repose sur l'intégration de plusieurs de ces infrastructures. En plus de cette intégration, il est nécessaire de définir des stratégies pour sélectionner la meilleure combinaison de ces infrastructures pour un contexte utilisateur donné. Dans la section suivante, nous analysons plusieurs systèmes adaptables intégrant certaines des infrastructures étudiées dans cette section et nous faisons ressortir les stratégies d'adaptation.

4. Exemples de systèmes adaptables

Les moyens décrits ci-dessus ont été expérimentés de façon indépendante. L'objectif de cette section est de décrire des systèmes adaptables qui intègrent plusieurs de ces moyens et d'étudier comment est réalisée cette intégration. Nous avons classé ces systèmes en trois catégories : les systèmes qui s'intéressent à l'adaptation du coté client, les systèmes dédiés à l'adaptation de documents HTML et les systèmes adaptables natifs, c'est-à-dire qu'ils ne reposent sur aucune infrastructure existante.

4.1. Navigateur

Une première catégorie de système adaptable résulte de l'intégration de techniques d'adaptation principalement dans le navigateur. Des exemples de tels systèmes sont nombreux.

Certains navigateurs proposent de créer un environnement auditif tridimensionnel correspondant à un document HTML [Mynatt 94, Kennel 96, Petrucci 00]. Cet environnement peut être accompagné d'une représentation tactile qui sert aussi de terminal d'entrée. Par exemple le système GUIB utilise un terminal d'entrée/sortie appelé GUIDE qui intègre un affichage de type braille. Ce terminal permet aux utilisateurs aveugles l'expérimentation de la manipulation directe combinée à la présentation audio. L'objectif de ces systèmes est de permettre à des personnes aveugles ou malvoyantes d'accéder à l'information disponible sur le Web.

WEST [Björk 99] et Powerbrowser [Buyukkokten 00] sont des navigateurs spécifiquement conçus pour être utilisés sur des assistants personnels. Ces navigateurs sont couplés à un service intermédiaire (proxy) qui a la charge de prendre une page HTML et de l'adapter à la navigation sur de petits écrans (cf. section suivante). De plus, ces navigateurs proposent une interface utilisateur pour visualiser les pages adaptées et faciliter l'interaction. Dans WEST, l'interface utilisateur repose sur la technique du flipZoom (cf. section 3.3). Dans

Powerbrowser, l'accent est porté sur l'aide à la navigation en offrant des fonctions pour ordonner les liens, pour filtrer les liens redondants, etc. Ces systèmes présupposent que le document HTML a déjà subi une réédition. Certains de ces systèmes proposent quelques solutions pour rééditer les documents HTML mais de façon relativement superficielle. Dans la section suivante, nous présentons quelques systèmes dédiés à ce besoin de réédition.

4.2. Réédition automatique de documents HTML

Un des problèmes qui se pose actuellement est le nombre toujours croissant de pages Web qui sont conçues pour des stations de travail et qui nécessitent de grands écrans et beaucoup de ressources. Cela a conduit à des travaux sur la réédition de documents HTML. L'objectif de la réédition de documents HTML est de permettre la réutilisation de ces pages dans le contexte d'adaptation. Nous présentons trois systèmes qui tentent de répondre à ce problème : Digestor, Aurora et FOM. D'autres systèmes similaires à ceux présentés peuvent être trouvés dans [Ma 00, Schilit 01].

4.2.1. Digestor

Digestor est un système qui réédite automatiquement n'importe quel document HTML afin de permettre de le visualiser de façon appropriée sur de petits écrans [Bickmore 99]. Autrement dit, Digestor découpe une page Web en plusieurs pages et ajoute des liens de navigation. Pour cela, le système Digestor s'appuie sur deux techniques d'adaptation étudiées précédemment : l'élision syntaxique (cf. section 3.1.5) et le transcodage (cf. section 3.1.3). À partir de ces techniques, une quinzaine de transformations ont été implantées : FullOutline, OutlineToH1, FirstSentenceElision, ReduceImage25%, etc. En combinant ces transformations, plusieurs solutions d'adaptation sont possibles. Par exemple, la figure 30 montre trois possibilités pour adapter une page HTML donnée.

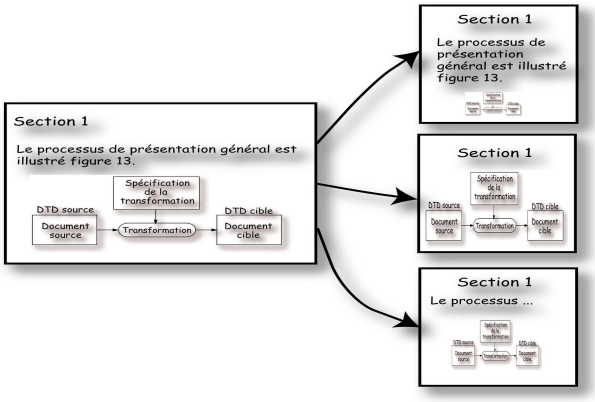


Figure 30. Différentes solution d'adaptation pouvant survenir dans Digestor.

Pour choisir quelles techniques employer et sur quelles parties du document HTML, Digestor recherche une séquence de transformation dans l'espace de combinaisons produit par ces transformations. La recherche s'effectue en utilisant une heuristique de type meilleur-d'abord [Pearl 90]. Chaque état dans l'espace de solution représente une version du document. L'état initial correspond au document initial. La fonction d'évaluation de l'heuristique pour chaque état est une estimation grossière de la taille d'écran nécessaire à l'affichage du document. Le passage d'un état à un autre s'effectue en appliquant une seule technique d'adaptation. À

chaque étape du processus de recherche, l'état le plus prometteur, c'est-à-dire celui qui nécessite le moins d'espace, est sélectionné et une technique d'adaptation est appliquée si possible. Aussitôt qu'un état est créé avec une version de document qui est « assez bonne », la recherche est arrêtée. Le document est jugé assez bon lorsque les pages générées peuvent être affichées entièrement sur l'écran. Le document correspondant est alors envoyé sur le terminal de l'utilisateur.

Digestor adresse uniquement le problème de la petite taille des écrans des assistants personnels ou autres terminaux. Pour fonctionner, l'utilisateur de Digestor doit saisir la taille de son écran.

4.2.2. Aurora

Le système adaptable Aurora [Huang 00] repose sur un modèle caractérisant le contenu d'une page Web. Ce modèle représente les tâches qu'un utilisateur doit accomplir à travers une collection de schémas. Chaque schéma spécifie les étapes concrètes que l'utilisateur doit effectuer pour accomplir une tâche donnée. En d'autres termes, il décrit le fonctionnement des objets à l'intérieur d'une page Web. Aurora s'appuie sur ce modèle pour adapter la page Web principalement en offrant des facilités de navigation.

Le processus de Aurora est constitué de deux composants. Premièrement, Aurora extrait les données d'un site Web et les transforme dans le schéma correspond. Cette transformation est effectuée par l'auteur en utilisant patML [PatML]. Ensuite, le document obtenu est adapté en utilisant des « adaptateurs » spécifiques, comme un adaptateur vers un document HTML purement textuel. Ces adaptateurs sont écrits en utilisant les outils disponibles pour transformer un document XML (CSS, XSLT, PatML, etc.).

L'avantage des schémas est que plusieurs sites ayant le même fonctionnement peuvent être convertis vers le même schéma.

Le principe d'Aurora est de laisser à l'auteur toutes les tâches à réaliser. L'extraction de la sémantique d'une page HTML est effectuée manuellement. Les adaptateurs sont aussi à développer.

4.2.3. FOM

Contrairement à Aurora, le système FOM [Chen 01] repose sur plusieurs techniques pour extraire automatiquement les intentions de l'auteur à partir du document de présentation au format HTML.

L'objectif de FOM est de comprendre les intentions de l'auteur en identifiant la fonction et la catégorie des objets d'un document HTML. La première étape consiste à identifier les objets du document HTML, selon qu'ils sont basiques ou composites. La reconnaissance des objets composites repose sur l'analyse de la mise en page des documents HTML. La reconnaissance de la fonction des objets repose sur un ensemble de règles. Ces règles visent à caractériser un objet basique ou composite selon un ensemble de propriétés, telles que la présentation, la sémantique, ou encore la décoration. La présentation caractérise la façon dont l'objet est représenté, c'est-à-dire son type de média, son format d'encodage et sa mise en page. La sémantique caractérise le contenu de l'objet. La décoration mesure le degré de décoration d'un objet. Le calcul de la valeur de la propriété de décoration repose sur des postulats du genre :

les objets Text/Video véhiculent généralement peu de décoration. Enfin la catégorie des objets est déterminée en fonction d'un arbre de décision.

À partir de ces objets, associés à une catégorie et une fonction, des règles d'adaptation sont appliquées. Par exemple, des objets ayant une haute valeur de décoration peuvent être filtrés. La décision repose sur un moteur de décision [Chen 00]. La décision s'effectue en quatre phases :

- décision à base de règle : cette phase décide quelle méthode peut ou doit être appliquée en comparant les objets du document avec le profil utilisateur et les capacités du terminal ;
- guide de prédiction et de préférence : cette phase affecte une priorité aux objets en accord avec les préférences de l'utilisateur et la probabilité sur la prédiction d'accès ;
- masque de mise en page : cette phase décide s'il existe une meilleure mise en page pour visualiser les objets. Ce choix repose sur un ensemble de mises en page prédéfinies ;
- compromis : cette phase sélectionne les objets finaux et les méthodes d'adaptation. Pour cela un compromis est effectué entre la qualité du contenu et le coût pour l'atteindre (le visualiser). La qualité se mesure par la quantité d'objets reçus par l'utilisateur et la qualité d'encodage du contenu. Le coût inclut la complexité de l'adaptation et le coût en temps que l'utilisateur doit dépenser.

Ce système est intéressant et semble donner de bons résultats. Cependant ces résultats dépendent de la puissance des techniques d'analyse sous-jacentes. Or, ces techniques ne peuvent pas donner de bon résultats sur des documents HTML dont la sémantique est implicite : elle est dans la mise en page.

4.3. Systèmes adaptables « natifs »

Contrairement à la catégorie précédente, les systèmes adaptables natifs ont une vision nouvelle de la publication des informations sur le Web. Ils proposent des solutions en adéquation avec le problème de l'adaptation sans tenir compte des pages Web existantes. Nous présentons deux systèmes : Avanti et Coocon.

4.3.1. Avanti

Le but du projet Avanti est de développer et d'évaluer un système d'informations distribué qui fournit des informations hypermédias sur des zones métropolitaines (restaurants, transports public, etc.) pour des utilisateurs différents (touristes, habitants, personnes aveugles, etc.) avec des intérêts, des connaissances et des capacités différents [Fink 98].

Les composants du système Avanti sont les suivants (cf. figure 31) :

- une base de données multimédia qui contient des informations sur le domaine d'Avanti encodées en plusieurs types de média ;
- un adaptateur d'hyper structure qui génère des descriptions du document hypermédia adapté ;

- un entrepôt de pages Web génériques (IRCS) qui inclut des règles d'exécution et des liens vers la base de données. Ces pages sont encodées en HTML étendu (cf. ci-dessous) ;
- un serveur de modèles utilisateur qui contient les caractéristiques de plusieurs utilisateurs. Ce serveur repose sur la définition de plusieurs stéréotypes liés au domaine de Avanti (cf. section 2.1.1) ;
- une interface utilisateur qui présente les informations hypermédia de façon adaptée.

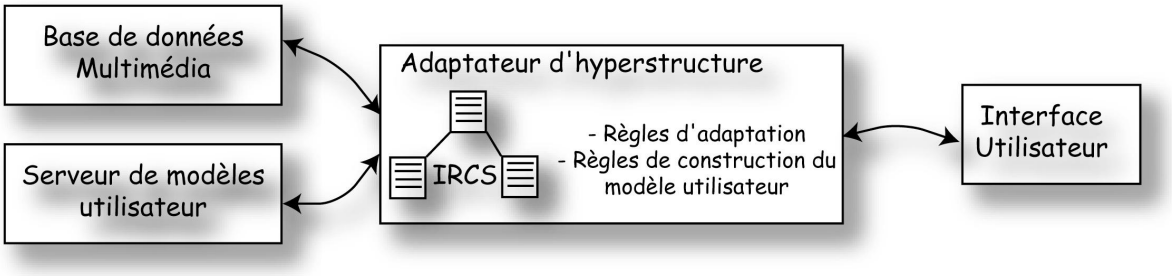


Figure 31. Architecture du système Avanti.

Les deux points intéressants de cette architecture sont l'adaptateur d'hyper structure et l'entrepôt de pages Web (IRCS). Les règles d'adaptation sont contenues dans les pages Web elles-mêmes. Dans Avanti, ces règles sont spécifiées par un auteur en utilisant le langage WebScripts [WebObject 01]. Par exemple, le code suivant est une règle d'adaptation :

```
1. <WEBOBJECT NAME="IfInterestInInformationForDisabled">
2.     <WEBOBJECT NAME=GoToAccessibilityInfo>
3.         <WEBOBJECT NAME=Accessibility.gif/>
4.     </WEBOBJECT>
5. </WEBOBJECT>
```

Avec les objets Web suivants :

```
1. IfInterestInInformationForDisabled:   WOConditionnal { condition=session.IIIIForDisabled }
2. GoToAccessibilityInfo:                WOHyperlink { pageName=@"AccessibilityInfo" }
3. Accessibility.gif :                   ImageTool { src="AccessibilityInfo.gif" }
```

On peut noter que c'est dans la fonction session.IIIIForDisabled que le lien est effectué avec le serveur de modèle utilisateur.

La force du système Avanti est de prendre en compte les caractérisques de l'utilisateur grâce à l'intégration d'un serveur de modèles utilisateur. Cependant, le langage WebScript ne permet en pratique que la mise en œuvre des techniques d'adaptation telles que l'alternative et le filtrage. De plus, la tâche d'adaptation incombe systématiquement à l'auteur du document. Par conséquent, son effort de conception se trouve accru de façon significative.

4.3.2. Cocoon

Cocoon est un cadre de travail pour la publication XML développé au sein de la fondation Apache [Cocoon]. Cocoon repose sur une séparation complète entre le contenu, la structure

logique et la présentation du document final. La production du document final repose sur le principe de pipeline. Ce pipeline suit globalement le processus figure 32. Les composants de ce processus sont les suivants :

- génération : la génération est le début du pipeline de production. Elle génère du contenu XML sous la forme d'événements SAX. Coocon fournit plusieurs générateurs : le générateur à partir d'un fichier, le générateur de la structure d'un répertoire local, etc. ;
- transformation : un transformateur génère des événements SAX à partir d'événement SAX. Les transformateurs de base sont XSLT, l'internationalisation, etc. ;
- sérialisation : la sérialisation est la fin du pipeline. Elle transforme les événements SAX en un flux de caractères ou un flux binaire. Le sérialiseur de base est la sérialisation vers HTML. Un service de sérialisation d'un document SVG transformé vers une image JPEG ou PNG est aussi fourni.

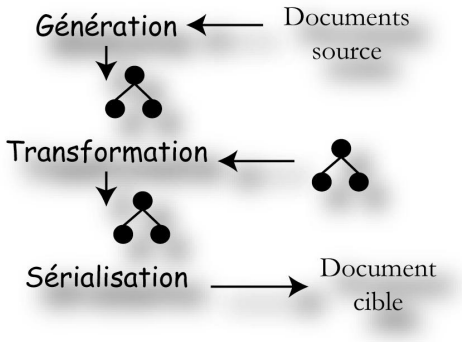


Figure 32. Processus général de Cocoon.

Les liens entre ces différents composants sont définis par l'auteur dans un fichier appelé *sitemap*. L'auteur peut ainsi choisir chacun des composants et les combiner à sa façon. Le problème avec cette approche est la complexité qui en résulte pour réaliser un site Web.

Cocoon n'est pas réellement un système adaptable, car il n'intègre pas la notion de contexte utilisateur. Néanmoins, Coocon est une implantation flexible et performante du principe de séparation du contenu et de la présentation.

5. Conclusion

Dans ce chapitre nous avons défini puis décrit les différents composants d'un système adaptable : le contexte de l'utilisateur et le système d'informations. Par ce dernier nous avons présenté les moyens existants pour adapter une présentation aux besoins et aux contraintes du contexte utilisateur. Nous avons ensuite décrit quelques systèmes adaptables existants. Cette description montre qu'il n'existe pour le moment aucun système adaptable satisfaisant pleinement les besoins de l'utilisateur.

Une des raisons à cela est que la réalisation d'un système adaptable implique le mariage de nombreux domaines de recherche : le domaine de l'intelligence artificielle, le domaine de la linguistique, le domaine de l'interface utilisateur, le domaine de l'hypertexte, le domaine du système et réseau et le domaine du génie documentaire. Dans chacun de ces domaines sont

proposées des solutions en accord avec leur domaine de compétence. Par exemple, le premier tente de développer des systèmes qui s'adaptent aux besoins de l'utilisateur selon ses capacités et ses connaissances. Le troisième domaine tente de résoudre le problème du faible espace de travail sur un écran (visualisation) et l'accès à l'information par des personnes handicapées (interaction et représentation sonore).

En résumé, un système adaptable doit répondre aux besoins suivants :

- il doit pouvoir maintenir une représentation du contexte de chaque utilisateur ;
- il doit intégrer plusieurs méthodes d'adaptation. Pour cela il est nécessaire de déterminer quand utiliser une méthode donnée, comment l'utiliser, quels sont les besoins de modélisation pour l'utiliser. De plus il est nécessaire de fournir un moyen pour contrôler le choix de la méthode à utiliser ;
- il doit fournir une interface utilisateur qui utilise des modalités d'entrée-sortie utilisables par des personnes handicapées de façon temporaire ou définitive.

Dans le chapitre suivant nous proposons quelques solutions pour répondre au deuxième besoin énoncé ci-dessus.

SECONDE PARTIE

CONTRIBUTION

Chapitre 4

Un système de production de documents multimédias adaptés au contexte de l'utilisateur

Dans ce chapitre, nous décrivons l'architecture et le processus du système adaptable que nous proposons. Pour chacun des composants de cette architecture, nous identifions les langages et les techniques permettant de réaliser l'adaptation, et nous montrons comment ces composants interagissent. Une partie de ce système a été validée à travers plusieurs expériences que nous relatons.

1. Introduction

Dans les deux chapitres précédents, nous avons étudié l'état d'avancement des travaux de recherche relatifs au génie documentaire en détaillant les travaux sur l'adaptation de document. De cette étude nous avons pu ressortir notamment qu'il n'existe pas pour le moment de solutions satisfaisantes pour présenter des documents multimédias appartenant à une classe, et qui prennent en compte les contraintes et les besoins induits par le contexte utilisateur. L'objectif de ce chapitre est de présenter notre contribution à ces deux besoins.

Le besoin de définir des classes de document est de faciliter la réutilisation des traitements. Ceux-ci, en reposant sur la classe du document et non sur l'instance, peuvent être réutilisés pour l'ensemble des documents appartenant à cette classe. Dans le contexte de la présentation de document, la conséquence directe est l'uniformité des présentations résultantes, sans pour autant être contraint par cette uniformité. Dans le chapitre 2, nous avons décrit les solutions existantes pour répondre au besoin de présenter des classes de documents dans le cadre de présentations statiques. Ces solutions reposent sur le principe fondamental de séparer les informations de contenu de la présentation. Cette séparation est essentielle dans le cadre de l'adaptation dont l'objectif est de produire plusieurs représentations d'un même contenu. C'est pourquoi notre approche repose sur ce principe.

Nous avons vu que la réalisation de la séparation des informations de contenu des informations de présentation nécessite un traitement pour les rassembler et ainsi produire la présentation finale. L'usage de feuilles de style ne permet pas de remettre fondamentalement en cause la structure du contenu (cf. chapitre 2 section 2.2.3.1) et donc ne répond clairement pas au besoin de produire des présentations multimédias complexes. Nous nous sommes tourné alors vers une approche par transformation qui permet de répondre aux besoins énoncés ci-dessus.

Dans la suite de ce chapitre, nous commençons par donner une vue d'ensemble de notre système adaptable à travers son architecture. Ensuite, nous identifions dans un premier temps les langages impliqués dans ce système, puis nous détaillons ses composants fonctionnels. Nous terminons ce chapitre en relatant les expérimentations que nous avons menées.

2. Le système adaptable

2.1. Cas d'étude : les documents de type SlideShow

Tout au long de ce chapitre, nous illustrons nos propos à travers un modèle de document utilisé comme cas d'étude. Ce modèle, appelé SlideShow, permet de représenter une suite de transparents. Nous présentons ce modèle à travers l'instance suivante :

```
1. <?xml version="1.0" encoding="ISO-8859-1"?>
2. <slideshow xmlns:db="http://www.oasis-open.org/docbook"
3.             xmlns:svg="http://www.w3.org/2000/svg">
4.    <header>
5.       <title>Edition de feuilles de transformation XSLT</title>
6.       <authors>
7.          <author presented="true">
8.             <firstname>Lionel</firstname>
9.             <surname>Villard</surname>
10.               <affiliation>
11.                  <orgname>Projet Opéra - INRIA Rhône-Alpes</orgname>
12.                  <address>
13.                     <street>655 avenue de l'Europe</street>
14.                     <postcode>38330</postcode>
15.                     <city>Montbonnot Saint-Martin</city>
16.                     <email>Lionel.Villard@inrialpes.fr</email>
17.                  </address>
18.               </affiliation>
19.            </author>
20.         </authors>
21.         <date day="7" month="11" year="2001"/>
22.         <video src="Edition.mpg"/>
23.      </header>
24.      <slides>
25.         <slide>
26.            <title>Plan</title>
27.            <block>
28.               <db:itemizedlist>
29.                  <listitem>
30.                     <para>But et méthodologie</para>
31.                  </listitem>
32.                  <listitem>
33.                     <para>Exemple SlideShow</para>
34.                  </listitem>
35.               …
36.               </db:itemizedlist>
37.            </block>
38.         </slide>
39.         <slide>
40.            <title>Exemple</title>
41.            <block>
42.               <svg:svg> … </svg:svg>
43.            </block>
44.         </slide>
45.      </slides>
```

```
46.  </slideshow>
```

Ce type de document permet de décrire une suite de transparents (slide). Il est composé un en-tête (header) qui contient le titre de la présentation, la liste des auteurs et la date de la présentation. Chaque auteur est caractérisé par un nom, un prénom et une affiliation. La vidéo du show est éventuellement représentée par l'élément video. Ensuite, la liste des transparents est décrite dans l'élément slides. Le contenu de chaque transparent (slide) est composé d'un titre (title) et de plusieurs blocs (block). Un bloc correspond à un regroupement de une ou plusieurs entités sémantiquement reliées. Le contenu de chaque bloc est une combinaison de plusieurs modèles de document : le modèle Docbook et le modèle SVG. En particulier, chaque bloc peut contenir une liste d'items (itemizedlist et listitem), des paragraphes (para) et des médias de type image, vidéo, audio et vectoriel en utilisant le modèle SVG. Ce modèle ne spécifie, à un certain niveau, aucune organisation ni dans le temps ni dans l'espace afin de ne pas restreindre les possibilités des présentations finales.

Pour chacun des documents appartenant au modèle sous-jacent, une ou plusieurs présentations peuvent être définies. La figure 33 montre une présentation de l'instance précédente. Cette présentation contient à gauche une table des matières contenant la liste des titres de transparent. À droite est représenté le contenu d'un transparent. Sur cette présentation, le passage d'un transparent à l'autre s'effectue de façon interactive en utilisant les boutons de navigation situés en haut à droite. La liste de auteurs, la date de la présentation et le titre de la présentation sont montrés dans la page de garde (non visualisées sur la figure).

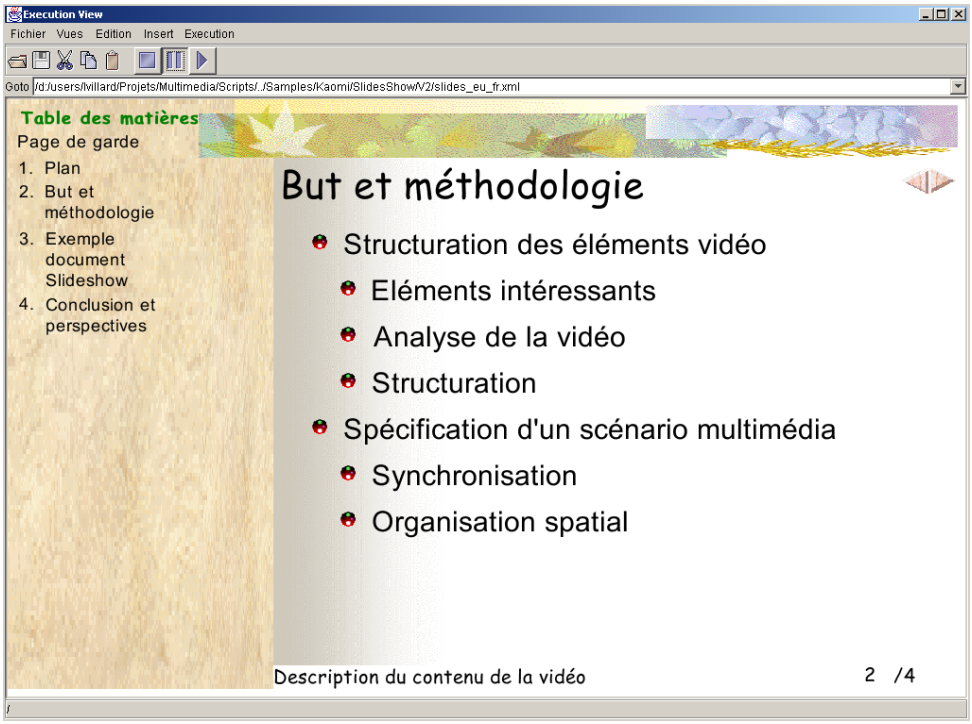


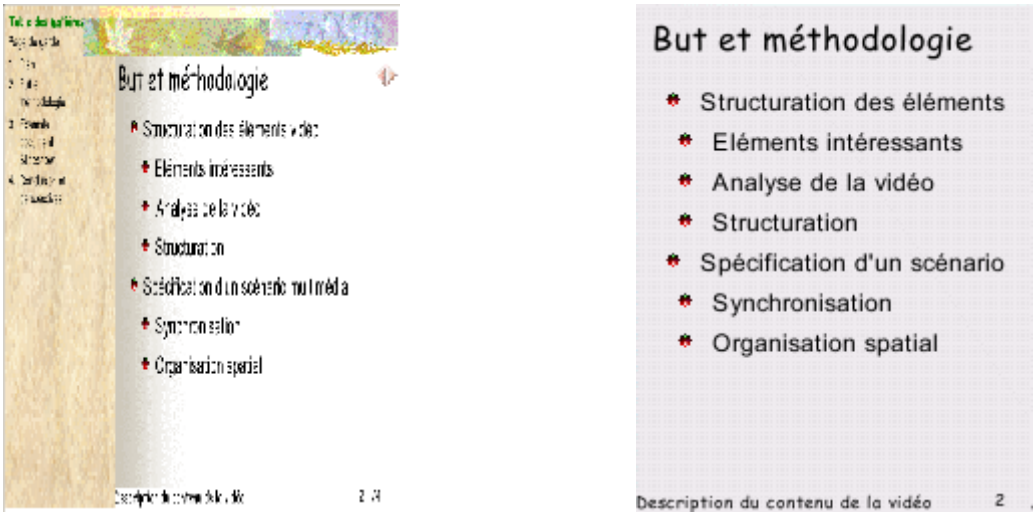
Figure 33. Présentation interactive d'une instance du modèle SlideShow.

Pour ce modèle de document particulier, nous avons identifié plusieurs paramètres d'adaptation. Chacun de ces paramètres reflète une préférence de l'utilisateur vis-à-vis de la présentation d'un SlideShow. Ces préférences sont les suivantes :

- le choix d'un style de présentation. Certaines personnes préfèrent une présentation sobre à l'opposé d'une présentation plus sophistiquée (fond d'écran, style graphique des bullets, etc.) ;
- le choix entre une présentation interactive ou non. Le passage de transparent en transparent peut être dirigé soit par l'utilisateur soit automatiquement ;
- le choix de montrer ou non la table des matières ;
- le choix entre une présentation en directe ou différée. Le même document peut être utilisé à la fois lors d'une présentation en direct devant un public, ou en différée dans un but pédagogique. Dans le premier cas, c'est l'orateur qui commente la présentation donc le document présenté ne contient ni bande son, ni vidéo de l'exposé. Dans le second cas, un enregistrement de l'orateur à travers une vidéo ou un son est diffusé en plus des autres médias.

Certains de ces choix doivent pouvoir être remis en cause durant la présentation par l'utilisateur, comme l'interactivité et la visualisation de la table des matières. Dans ce cas, l'adaptation s'effectue dynamiquement pendant l'exécution de la présentation multimédia.

En plus de ces paramètres, nous considérons quelques uns des paramètres d'adaptation de présentation étudiés dans le chapitre précédent. Ces paramètres sont les formats de médias et les modèles de document supportés par le terminal de restitution ainsi que sa taille d'écran. Une adaptation simple pour montrer la présentation illustrée figure 33 sur un PDA est de générer une image dans un format supporté par le PDA, de la réduire en utilisant la technique de zoom afin qu'elle soit visualisable entièrement sur l'écran et de la restituer sur le PDA (cf. figure 34a). Le résultat de cette opération rend clairement le document inexploitable car les caractères sont tellement petits qu'ils sont devenus illisibles. Par contre la présentation de la figure 34b, qui est nettement plus lisible, correspond mieux aux attentes d'un système adaptable vis-à-vis du résultat qu'il peut produire.



(a) (b)

Figure 34. Présentation SlideShow sur un PDA. À gauche est utilisée la technique de Zoom. À droite est montrée une présentation souhaitée.

Notre objectif est que le système adaptable puisse à partir des mêmes modèles de document source et des mêmes instances produire automatiquement les présentations multimédias de la figure 33 et de la figure 34b.

2.2. Architecture et processus d'adaptation

La figure 35 illustre l'architecture de notre système adaptable. Le document source contient de façon structurée les informations de contenu nécessaires pour la présentation finale. Cette présentation est obtenue en appliquant successivement trois processus indispensables :

- la **transformation** : ce processus permet de convertir les informations logiques contenues dans le document source vers une représentation graphique. Pour simplifier l'écriture des feuilles de transformation, un document de présentation de haut niveau est introduit entre la transformation et l'exécution ;
- le **formatage** : ce document de présentation de haut niveau est *formaté* par le processus de formatage afin de produire les structures nécessaires pour être directement exécutable. Il faut noter que le formatage ne peut pas directement manipuler le document source car c'est un processus spécifique qui ne connaît pas *a priori* le modèle du document source et donc ne peut pas associer une présentation à son contenu ;
- l'**exécution** : comme dans tout système de présentation multimédia, il est nécessaire d'avoir un processus pour ordonnancer la présentation : c'est le rôle du processus d'exécution. Il repose sur des informations de bas niveau contenues dans le document formaté.

Au niveau de chacun des composants (transformation, formatage et exécution) de cette architecture un ensemble de techniques d'adaptation est appliqué. Ces techniques peuvent soit être intégrées au sein même du composant ou faire appel à un *service d'adaptation*. En particulier, ces services incluent un service de transcodage de médias et un service de modélisation du contexte de l'utilisateur. Pour ce dernier service, nous intégrons la description du profil des terminaux au sein du GUMS (cf. section 2.1.2 du chapitre 3).

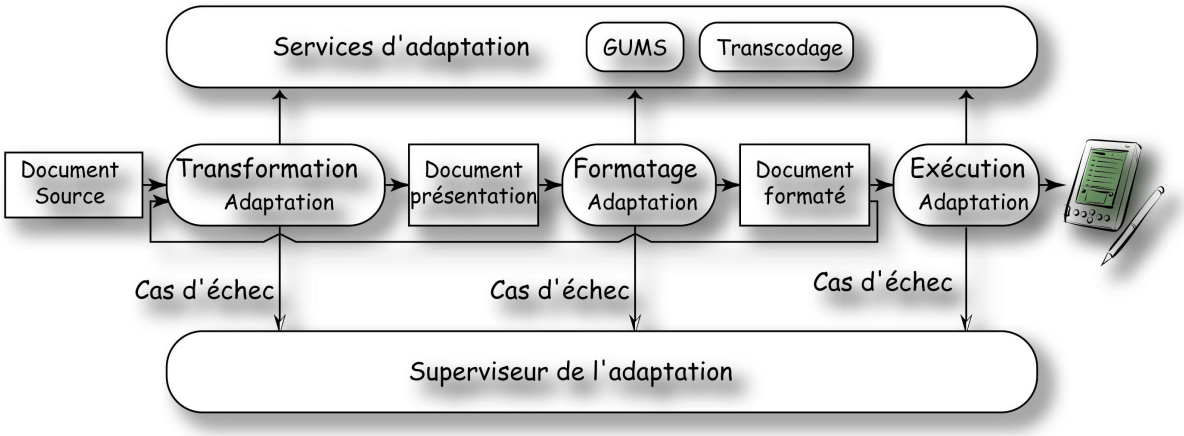


Figure 35. Architecture générale du processus d'adaptation.

L'accès aux documents au moyen de ce système peut aboutir à des cas d'échec. Par exemple, lors de l'exécution d'une vidéo, un cas d'échec peut être l'impossibilité de réduire le nombre

d'images à cause d'une trop forte dégradation de la qualité. Pour cela, nous introduisons un superviseur d'adaptation qui s'occupe des cas d'échec.

Le module de transformation fait intervenir plusieurs modèles de document : le modèle source, le modèle de transformation et le modèle de présentation (cf. figure 36). L'usage d'un processus de transformation permet de conserver une grande latitude sur le choix du modèle du document source. Aucune restriction n'est donc posée à ce niveau sur ce modèle, hormis le fait qu'il doit s'appliquer à des documents XML. Dans notre cas d'étude, ce modèle est le modèle SlideShow. Par contre, le choix et/ou la définition du langage pour spécifier la transformation et pour décrire des présentations multimédias (document cible) nécessite de définir plus précisément les besoins requis pour atteindre nos objectifs. Nous verrons que cette étude va nous conduire aux langages XSLT et Madeus 2.0.

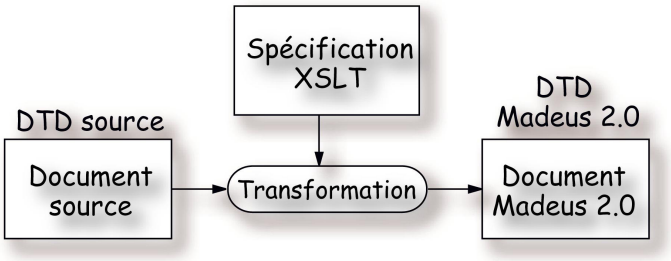


Figure 36. Langages impliqués dans le processus de transformation.

Le principal apport de ce chapitre est la conception et la mise en œuvre d'une infrastructure permettant l'adaptation des documents multimédias. Nos contributions portent d'une part sur la définition et/ou les choix des langages intervenant au niveau de la transformation puis sur la mise en adéquation des langages choisis. D'autre part, elles portent aussi sur la description précise du rôle des composants (transformation, formatage, exécution) de cette architecture et de leurs extensions, entre autres, par les techniques étudiées dans le chapitre précédent, pour prendre en compte les besoins d'adaptation. Ce dernier ensemble de contributions est décrit dans la section 4.

3. Langages pour la transformation et la présentation

Dans cette section, nous identifions les besoins en terme de transformation puis nous analysons et choisissons un langage de transformation répondant à ces besoins. Nous effectuons ensuite le même travail concernant le langage pour représenter les documents multimédias.

3.1. Langage de transformation

3.1.1. Identification des besoins

Le langage de transformation, et donc le processus de transformation sous-jacent, doivent répondre aux besoins suivants :

- il doit pouvoir s'appliquer à des documents XML, puisque le format du document source est XML ;

- il doit permettre de spécifier des transformations de base, comme la suppression et la création de nœuds, la conversion d'un attribut vers un élément, etc. Par exemple, les titres de chaque transparent doivent être facilement transformés en objets de type texte ;

- il doit permettre des transformations plus complexes comme la réorganisation des nœuds. Nous avons vu en effet que la structure logique d'un document ne suit pas toujours sa structure spatio-temporelle. Par exemple, dans la présentation interactive d'un document SlideShow (cf. figure 33), le titre de la présentation est présenté au dessous des transparents, alors qu'il est défini avant dans le document source ;

- il doit pouvoir générer plusieurs éléments cible à partir d'un même élément source. En effet, le document cible est caractérisé par les multiples dimensions qui composent tout document multimédia. Pour un même nœud source, il est nécessaire de générer à la fois son placement spatial et son placement temporel. De plus un même noud source peut apparaître plusieurs fois, comme le titre d'un transparent ;

- le dernier besoin est que le langage de transformation doit être suffisamment simple pour faciliter son édition, tout en satisfaisant les besoins cités précédemment.

3.1.2. Choix du langage

Dans la section 3.2.1 du chapitre 2, nous avons cité les principaux langages de transformation existant pour les documents. Dans cette section, nous classons ces langages afin d'en faire ressortir les caractéristiques et de montrer leur adéquation vis-à-vis des besoins énoncés ci-dessus. De plus, nous élargissons notre champ d'étude aux langages de programmation ce qui nous amène à l'identification de trois catégories de langages pour décrire des transformations. La première catégorie de langages inclut les langages de programmation (Java, C, etc.) et les scripts (Tcl, Perl, etc.). Ces langages sont génériques dans le sens qu'ils ne sont pas dédiés à la transformation de documents. Certes, ils sont expressifs mais ils sont difficiles à utiliser et ils ne sont pas adaptés pour permettre une édition interactive.

La deuxième catégorie de langage regroupe les langages de programmation intégrant des fonctions de transformation. C'est le cas, en particulier, de Omnimark [Omnimark5], de Balise [Balise] et de XML Script [<XML>Script]. Bien que ces langages facilitent la conception de transformations, ce sont des langages impératifs et par conséquent leur utilisation reste relativement complexe et ils ne peuvent pas être intégrés dans un système d'édition convivial.

Enfin la dernière catégorie inclut les langages dédiés exclusivement à la transformation, comme le langage de transformation d'Amaya [Amaya], le langage Chameleon [Mamrak 89] et le langage XSLT [XSLT 99]. La majorité des langages de cette catégorie s'applique à des documents SGML et ne couvre qu'un sous-ensemble des besoins d'expressivité énoncés dans la section précédente. Par exemple, le langage de transformation d'Amaya ne permet par de réordonner les éléments [Bonhomme 98]. L'exception est le langage XSLT qui s'applique à des documents XML. De plus, il regroupe dans un même langage plusieurs traits existants dans les autres langages. Entre autres, la transformation dans XSLT est à la fois dirigée par la source et effectuée par requête (cf. chapitre 2 section 3.2.1.1). De plus le mode de génération du résultat peut être linéaire (flot de caractères) ou arborescent. La conception de XSLT repose sur les concepts des langages de programmation fonctionnelle. Les règles de transformations sont des fonctions pures : elles définissent un fragment de l'arbre cible en

fonction d'un fragment de l'arbre source et ne produisent aucun effet de bord. En théorie « si un langage est sans effet de bord alors lorsqu'une modification est effectuée sur le document source, il est possible de calculer les changements à appliquer sur le document cible sans être contraint de transformer une nouvelle fois le document source à partir de zéro » [Kay 00]. Ce langage est donc très intéressant du point de vue éditorial. Pour toutes ces raisons, notre choix s'est orienté vers ce langage.

Cependant XSLT souffre de quelques limitations relativement gênantes lorsque le document cible est de nature multimédia. Notamment un des problèmes de XSLT est l'impossibilité de désigner des éléments générés lors de l'exécution de la transformation. Par conséquent, il est difficile de spécifier des références croisées entre les dimensions du document. En effet, les langages de présentation actuels reposent sur une désignation explicite portant sur l'identificateur de l'élément pour effectuer des références croisées. Dans SMIL 2.0, l'association d'un objet média avec une région s'effectue en spécifiant l'identificateur de la région, comme le montre l'exemple suivant :

```
1.  <smil>
2.      …
3.      <region id="region-video" …/>
4.      …
5.      <video region="region-video" …/>
6.      …
7.  </smil>
```

Dans cet exemple, l'identificateur de la région est utilisé comme valeur de l'attribut region de l'élément video. Pour effectuer des références croisées, l'auteur doit donc s'assurer que l'identificateur source (porté par l'élément region) soit identique à l'identificateur cible (porté par l'élément video). Lorsqu'un tel document est généré par transformation, les identificateurs sont générés dynamiquement et donc l'auteur doit respecter une politique de nommage pour générer des identificateurs identiques à des moments différents. Or cette contrainte peut complexifier grandement la conception d'une feuille de transformation. Pour s'en convaincre, prenons l'exemple suivant :

```
1.  <seq>
2.      <xsl:apply-templates select="*">
3.          <xsl:sort …/>
4.      </xsl:apply-templates>
5.  </seq>
```

Cet exemple est un fragment d'une transformation XSLT qui, une fois exécuté, génère une séquence temporelle (élément seq) de plusieurs médias. Ces médias sont, dans cet exemple, générés lors de l'exécution de l'instruction XSLT apply-templates. Cette instruction sélectionne un nombre indéterminé de nœuds du document source, trie ces nœuds (instruction xsl:sort) et recherche une règle de transformation pour chacun de ces nœuds. Ces règles, qui ne sont pas spécifiées ici, génèrent un ou plusieurs médias. Pour résumé, cette transformation génère un certain nombre de médias dont le nombre, le type et l'ordre de génération sont inconnus.

Supposons qu'à partir de cette spécification, l'auteur veuille afficher une image pour indiquer qu'il ne reste plus que deux médias à présenter. Pour cela, il doit connaître l'identificateur de

l'avant-dernier média généré pour les raisons évoquées ci-dessus. Il doit donc parcourir l'ensemble des règles générant les médias, analyser la politique de nommage du média et espérer que chaque règle emploie cette même politique, ce qui n'est pas toujours le cas surtout lorsque des fragments de feuille de transformation sont réutilisés. Pour un besoin de spécification très simple, sa mise en œuvre peut devenir relativement compliquée.

Pour répondre à ce besoin de désignation, nous avons étudié plusieurs solutions dont celle qui consiste à étendre le langage de transformation. Nous avons testé l'ajout d'une nouvelle fonction, appelée current-target, qui permet de récupérer le nœud courant dans l'arbre cible. La solution du problème précédemment peut s'exprimer alors de cette façon :

```
1.  <seq>
2.      <xsl:apply-templates select="*">
3.          <xsl:sort …/>
4.      </xsl:apply-templates>
5.      <image id="Warning" begin="{current-target()/*[last() - 2]/@id}"/>
6.  </seq>
```

Dans cette transformation, l'image Warning est synchronisée avec l'avant dernier média grâce à la fonction current-target. En effet, l'élément cible courant est, dans ce cas, le résultat de l'exécution de l'élément seq qui contient les médias générés. L'expression current-target()/*[last() - 2]/@id sélectionne à partir de cet élément (current-target) l'avant dernier fils (*[last() - 2]*) et extrait la valeur de l'attribut id (@id). Comme on le voit, cette solution permet de répondre simplement à notre problème. Cependant, considérons le cas de transformation suivant :

```
1.  <for-each select="current-target()/*">
2.      <image …/>
3.  </for-each>
```

Dans cet exemple, une image est générée pour chacun (for-each) des fils du nœud cible courant (current-target()/*). Or l'image générée est elle-même un fils de ce noeud. En plus de rompre le principe fondamental de XSLT qui est le fait de demeurer sans effet de bord, cette boucle est infinie. Cette solution est donc inadaptée à nos besoins.

Une autre solution plus simple est d'étendre le langage cible par un mécanisme de désignation relative. Dans la spécification suivante, l'image Warning est synchronisée avec l'avant dernier média qui est positionné au même niveau hiérarchique que l'image (ils sont donc voisins) mais deux positions avant. En utilisant la syntaxe XPath, cela se spécifie par l'expression preceding-sibling[2]. La solution à notre problème s'exprime donc comme suit :

```
1.  <seq>
2.      <xsl:apply-templates/>
3.          <xsl:sort …/>
4.      </xsl:apply-templates>
5.      <image begin="preceding-sibling[2]" id="Warning"/>
6.  </seq>
```

Cette solution a l'avantage de simplifier la transformation tout en évitant les effets de bord de la première solution et la remise en cause de la syntaxe XSLT. Nous verrons plus en détails cette solution dans la section 3.2.7.

3.2. Modèle de document de présentation multimédia Madeus 2.0

3.2.1. Identification des besoins

Dans le second chapitre nous avons décrit les principaux modèles de documents multimédias existants. Le point commun de ces langages est qu'ils ont été conçus pour être utilisés directement par un auteur. Dans un contexte de génération, ces langages souffrent de certains manques qui se traduisent principalement par une spécification complexe de la transformation. Les exemples ci-dessous montrent quelles sont les lacunes de chacun des langages suivants : SMIL 2.0, ZYX, Madeus 1.0 et XSL-FO.

Le point faible du langage SMIL 2.0, et du modèle ZYX, sont leur pauvreté au niveau du positionnement des médias dans la dimension spatiale. En effet, seul un positionnement absolu est possible. Cette lacune ne pénalise pas uniquement le processus de génération, mais de façon plus large restreint le pouvoir d'expression d'une présentation multimédia. L'exemple classique est le positionnement d'objets texte les uns en dessous des autres. Au moment de l'écriture ou de la génération de la présentation, la dimension de chaque objet texte est inconnue.

Le langage Madeus 1.0 définit une seule structure pour spécifier à la fois la structure logique, spatiale et temporelle de la présentation, ce qui restreint son pouvoir d'expression. De plus, Madeus 1.0 ne propose pas d'opérateur (équivalent au **seq** et au **par** de SMIL) offrant une stratégie de placement des objets contenus dans celui-ci. Or ces opérateurs permettent de simplifier sensiblement l'écriture de feuilles de transformation. Par exemple, le fragment de la feuille de transformation qui permet de spécifier que plusieurs transparents se jouent en séquence est le suivant :

```
1.  <xsl:template match="slides">
2.      <!-- Génère les intervalles pour chaque transparent -->
3.      <xsl:apply-templates select="slide"/>
4.      <!-- Génère les relations temporelles -->
5.      <xsl:for-each select="slide">
6.          <xsl:if test="position() > 1">
7.              <madeus:meets interval1="slide{position() - 1}"
                              interval2="slide{position()}"/>
8.          </xsl:if>
9.      </xsl:for-each>
10. </xsl:template>
```

Dans cet exemple, il est nécessaire de positionner pour chaque objet une relation **meets** avec l'objet qui le précède structurellement. Il apparaît clairement que l'usage d'un opérateur **seq** simplifie grandement cette feuille de transformation en supprimant les lignes 5 à 9 :

```
1.  <xsl:template match="slides">
2.      <!—Génère les intervalles pour chaque transparent -->
3.      <seq>
4.          <xsl:apply-templates select="slide"/>
```

```
5.        </seq>
6.    </xsl:template>
```

Le dernier langage, XSL-FO [XSL 01], permet de décrire la pagination et le style d'une information de contenu. La mise en page repose sur un modèle de boîtes organisées en séquence et réparties à travers une ou plusieurs pages. La description du style est issue principalement de CSS2. XSL-FO n'est pas un langage multimédia car il ne permet que d'organiser les médias dans la dimension temporelle. Il pourrait cependant servir de modèle spatial notamment pour combler les lacunes de SMIL 2.0 dans cette dimension. Néanmoins, cette solution n'est pas concevable car la mise en page d'une présentation multimédia n'est ni stricte (la notion de page en multimédia n'existe pas) ni statique comme celle définie dans XSL-FO. Les documents XSL-FO ne sont donc pas flexibles ce qui limite son champ d'adaptation. Seul le principe d'emboîtement est repris dans notre modèle, en particulier à travers l'élément S-Group.

Enfin la dernière lacune de ces langages vis-à-vis de la génération est leur mécanisme de désignation. Seule la désignation par le nom explicite de l'élément est permise. Nous avons montré dans la section précédente la nécessité d'étendre ce mécanisme de désignation.

Notre démarche pour décrire notre modèle, appelé *Madeus 2.0* (car très fortement issu de Madeus 1.0), s'inspire de la tendance actuelle qui consiste en la définition de *modules*. Certains de ces modules sont regroupés pour constituer une *fonctionnalité* du modèle. Certains de ces modules sont issus des travaux de modélisation effectués dans Madeus 1.0 [Layaïda 97]. D'autres modules sont équivalents à ceux définis par le langage SMIL 2.0. Enfin, plusieurs modules ont été définis spécifiquement pour répondre au besoin d'adaptation.

Le but de cette description est de donner un aperçu des principaux constituants de Madeus 2.0. Nous ne détaillons pas la sémantique précise de chaque élément car cela dépasserait largement le cadre de cette thèse. Nous nous focalisons sur les modules qui sont nouveaux par rapport à ceux définis par Madeus 1.0 et SMIL 2.0. Cependant, pour les situer, il est nécessaire de décrire rapidement les modules qui les utilisent.

3.2.2. Module structure générale

Dans Madeus 2.0, la description d'un document multimédia est organisée en quatre dimensions : la dimension logique, la dimension spatiale, la dimension temporelle et la dimension hypermédia. En accord avec ce découpage, trois structures distinctes sont définies pour représenter les trois premières dimensions. Un document Madeus 2.0 ressemble donc à ceci :

```
1.    <madeus>
2.        <actor> … </actor>
3.        <spatial> … </spatial>
4.        <temporal> … </temporal>
5.    </madeus>
```

L'élément actor regroupe la définition des objets médias de la présentation multimédia. L'élément temporal permet de spécifier la synchronisation temporelle entre ces objets. Le dernier élément spatial regroupe l'organisation spatiale des objets de la présentation. Le

contenu de ces trois éléments est décrit dans les trois sections suivantes. La dimension hypermédia est spécifiée à travers un jeu d'attributs que nous définissons dans la section 3.2.6.

3.2.3. Fonctionnalité acteur

Cette fonctionnalité permet de décrire les objets, que nous appelons *acteurs*, qui sont joués pendant la présentation du document. Ces objets sont définis indépendamment de leur position sur l'écran et de leur intervalle de temps. La description des acteurs est répartie en trois modules : le module *média*, le module *composition d'acteurs* et le module *style*.

Le module *média* est équivalent au module du même nom défini dans SMIL 2.0. Il définit plusieurs éléments représentant les médias de la présentation. Ces éléments sont text, image, video, audio, animation et ref.

Le module *composition d'acteurs* définit un élément a-group qui permet d'organiser de façon hiérarchique les médias. Aucune sémantique de présentation n'est associée à cet élément. L'auteur peut l'utiliser à son gré pour l'organisation des médias.

Sur chacun des éléments définis dans les deux modules précédents peuvent être associées plusieurs informations de *style* en utilisant l'attribut style. Par exemple, pour un objet texte, les styles associés peuvent être la couleur, la taille de police de caractère, etc. Lorsque des attributs de style sont spécifiés sur un élément défini dans le module de composition, la valeur de ces attributs est propagée sur les descendants de l'élément correspondant. Cette valeur constitue alors la valeur par défaut, qui peut éventuellement être surchargée par chacun des descendants en spécifiant une autre valeur. La syntaxe de définition des propriétés de style est celle définie par CSS.

Par exemple, le document suivant est un extrait de la présentation de la table des matières du document SlideShow.

```
1.  <a-group id="table of content">
2.  <image id="A-tocback" src="motif.gif"/>
3.  …
4.  <text id="A-tocTitle" style="font-size:12; font-family: Comic Sans MS">
5.     Table des matières
6.  </text>
7.  …
8.  </a-group>
```

3.2.4. Fonctionnalité synchronisation temporelle

Cette fonctionnalité définit les éléments nécessaires à l'organisation à travers le temps des acteurs. Elle est le résultat de la fusion de Madeus 1.0, concernant la spécification de relations temporelles, et de SMIL 2.0 principalement par l'usage d'opérateurs temporels. Nous définissons quatre modules : le module *intervalle*, le module *composition temporelle*, le module *relation temporelle* et le module *lien* avec un acteur.

Le module *intervalle* permet de définir des intervalles de temps caractérisés par un instant de début, une durée et un instant de fin. Un intervalle est défini par l'élément interval sur lequel les attributs begin, end et duration peuvent être spécifiés. La valeur de ces attributs est un intervalle spécifié par trois valeurs, la borne minimum, la borne maximum et la valeur

préférée. Par exemple, la spécification duration="min:12s pref:15s max:20s" signifie que la durée de l'intervalle doit être comprise entre 12 et 20 secondes, et que la durée préférée est 15s.

Le module *composition temporelle* définit plusieurs éléments composites (ou opérateurs) portant une sémantique temporelle particulière. Ces opérateurs sont les trois opérateurs seq, par et excl issus du langage SMIL 2.0.

Tandis que le module précédent permet de spécifier à gros grain la synchronisation entre les acteurs, le module *relation temporelle* permet de raffiner cette synchronisation grâce à des relations temporelles. Ces relations permettent de positionner temporellement deux éléments de façon relative. Par exemple la spécification A before B spécifie que l'objet A doit se jouer avant l'objet B. Ces relations, qui existent dans Madeus 1.0, sont celles définies par Allen [Allen 83] auxquelles sont ajoutées des informations quantitatives et des relations causales [Layaïda 97].

Enfin le dernier module permet d'associer un intervalle de temps à un acteur. L'attribut actor, positionné sur un intervalle, contient l'identificateur de l'acteur associé. Lorsque l'intervalle spécifie une durée pour le média, alors le comportement à l'exécution dépend de la durée intrinsèque portée par le média. Si l'intervalle de durée est plus grand que la durée intrinsèque, alors l'attribut additionnel fillActor spécifie le comportement désiré selon sa valeur :

- repeat : le média est rejoué pendant toute la durée de l'intervalle ;
- freeze : le média reste affiché jusqu'à la fin de l'intervalle. Lorsque le média est une vidéo, la dernière image est affichée ;
- cut : le média est effacé de la présentation avant la fin de l'intervalle.

Dans le cas contraire, c'est-à-dire lorsque la durée de l'intervalle est plus petite que la durée intrinsèque, le média est interrompu.

Par exemple, un fragment de la partie temporelle de la présentation du document SlideShow illustrée figure 33 est le suivant :

```
1.    <par id="T-pgarde" duration="pref:10s">
2.        <interval actor="A-pgardeTitle" Duration="pref:10s max:15s">
3.        <seq>
4.        …
5.        </seq>
6.    </par>
```

La fonctionnalité temporelle, telle qu'elle est décrite dans cette section, est un mixte entre Madeus 1.0 pour l'aspect relationnel et flexible, et SMIL 2.0 pour l'expressivité.

3.2.5. Fonctionnalité placement spatial

La fonctionnalité placement spatial permet d'organiser les médias dans la dimension spatiale. Cette fonctionnalité est similaire à la précédente. Elle contient quatre modules : le module région, le module composition spatiale, le module relation spatiale et le module lien avec un acteur.

Le module *région* permet de décrire une région spatiale en deux dimensions grâce à l'élément region. À cet élément sont associés cinq attributs left, top, right, bottom et depth, correspondant aux dimensions de la région et à sa profondeur dans les plans. La valeur de ces attributs est un intervalle de valeurs comme pour l'attribut temporel duration.

Le module *composition spatiale* permet de regrouper plusieurs régions. À la différence de SMIL, Madeus 2.0 définit plusieurs opérateurs auxquels est associée une politique de placement des fils contenus dans l'opérateur. Par exemple, l'opérateur flow permet de positionner les éléments de bas en haut, de haut en bas, de gauche à droite ou de droite à gauche selon la valeur de l'attribut direction. L'espacement entre les éléments est spécifié en utilisant l'attribut space contenant un intervalle de valeurs.

De la même façon que Madeus 2.0 définit un jeu de relations temporelles, plusieurs relations spatiales sont définies dans les deux dimensions. Ces relations, issues de Madeus 1.0, sont par exemple leftAlign qui permet d'aligner deux objets selon leur bord gauche, topSpacing qui permet d'espacer deux objets selon leur bord supérieur, etc.

Enfin le dernier module permet d'associer une région à un acteur grâce à l'attribut actor. L'attribut fit permet de contrôler la relation entre la dimension intrinsèque de l'acteur et sa dimension dans la région. Les valeurs possibles de cet attribut sont celles définies par le langage SMIL 2.0. Par exemple, la valeur fill signifie que l'objet est retaillé pour correspondre aux dimensions de la région.

Par exemple :

```
1.  <region width="800" height="600">
2.      <region actor="A-tocback"/>
3.      <flow direction="vertical" width="200" left="10" top="10" depth="21">
4.          <region ID="R-TOC_title" Actor="A-TOC_title"/>
5.          <region ID="R-TOC_title" Actor="A-guard"/>
6.          …
7.      </flow>
8.  </region>
```

3.2.6. Fonctionnalité lien

Le but de cette fonctionnalité est de décrire les liens de navigation entre les éléments du document. La base de cette fonctionnalité est le langage XLink [XLink 01] que nous avons étendu pour décrire des comportements temporels et spatiaux. Les propriétés supportées par le module de lien sont les suivantes :

- Comportement d'affichage : lorsqu'un média est activé, l'élément cible peut être soit affiché dans une nouvelle fenêtre, soit remplacé par le média source ou soit inclut dans la présentation courante.

- Localisation de la cible : cette propriété permet de spécifier la localisation de la cible du lien. Dans XLink, cette cible est une URL. Cette URL est étendue pour prendre en compte le besoin de localiser un instant de temps particulier de la présentation et une région précise. La localisation temporelle peut être soit absolue (une date), soit relative par rapport à un élément du scénario temporel. De la même façon, la localisation spatiale peut être soit absolue, soit relative.

- Type d'activation : cette propriété définit comment le lien doit être activé. Trois attributs permettent de caractériser cette propriété :
    - l'attribut hrefDur définit la période de temps pendant laquelle le lien peut être activé ;
    - l'attribut hrefNumber définit le nombre de fois que le lien peut être activé ;
    - et l'attribut hrefActivation définit le moyen utilisé pour activer le lien. Il peut être automatique ou manuelle. Ce dernier moyen implique que l'utilisateur utilise une modalité d'entrée.

Par exemple, la spécification suivante décrit un lien temporel entre la première entrée de la table des matières et le groupe temporel spécifiant la présentation du premier transparent. Le lien temporel est spécifié grâce à la commande T-slide1.begin représentant la date de début du groupe temporel T-slide1.

```
1.    …
2. <text id="slide1_toc" href="T-slide1.begin" hrefActivation="mouse.OnClick">
3.    Plan
4. </text>
5. …
6. <par id="T-slide1">
7.    …
8. </par>
```

3.2.7. Module désignation relative : XPath

Un des besoins avec l'utilisation des mécanismes de transformation est l'usage de désignation relative pour simplifier la spécification de la transformation. En effet, pour positionner une relation ou un évènement entre deux médias, il est nécessaire de connaître l'identificateur de ces médias. Or souvent dans un processus de transformation, l'identificateur n'est pas connu puisqu'il est généré (cf. section 3.1.2).

Nous proposons d'utiliser le langage XPath pour désigner les éléments de façon relative. XPath est un langage très complet, qui permet d'exprimer des expressions booléennes, numériques et des sélections d'un ensemble de nœuds (cf. chapitre 5, section 4.4.1). Dans notre contexte de génération de présentation Madeus 2.0, seule la sélection d'un nœud nous intéresse. Par conséquent, nous imposons quelques restrictions d'usage et précisons la sémantique lors de la sélection de plusieurs nœuds :

- seules les expressions de sélection de noeuds peuvent être spécifiées. Par exemple, l'expression count(section), qui compte le nombre d'éléments de type section, n'est pas valide car son résultat est un nombre. Par contre, l'expression section[position() < count(block)] est valide car son résultat est l'ensemble des éléments de type section telle que leur position est inférieure au nombre d'éléments de type block ;
- lorsque plusieurs nœuds sont sélectionnés, seul le premier est utilisé ;
- lorsque aucun nœud n'est sélectionné, l'attribut associé est ignoré.

Nous avons choisi d'autoriser la spécification de sélection de plusieurs nœuds, parce qu'il n'est pas possible de vérifier statiquement quand une expression sélectionne zéro, un ou plusieurs nœuds. Pour avoir un comportement déterministe lors du formatage, nous avons donc choisi de préciser un comportement pour chacun des cas de sélection, plutôt que de signaler une erreur. L'auteur ne peut en effet pas tester tous les cas de génération et l'utilisateur ne doit pas être confronté à des erreurs de formatage.

3.2.8. Bilan sur le modèle de présentation proposé

Un des critères d'évaluation d'un modèle de document de présentation est sa capacité à permettre la réutilisation du contenu d'un document multimédia. Dans notre modèle, seul le document entier peut être réutilisé. En effet, la réutilisation d'un fragment de la présentation est difficile à cause de l'éclatement du scénario en plusieurs structures. Cependant, ce besoin de réutilisation n'est pas fondamental dans un contexte de génération pour lequel, le problème est de réutiliser *la spécification de la transformation* plutôt que le contenu du document.

Le modèle de document que nous avons présenté dans cette section n'a de sens qu'à condition de pouvoir le formater et le jouer. La difficulté majeure est la prise en compte des intervalles de valeurs, ainsi que les relations spatiales et temporelles. Cependant, il a été montré que ce type de fonctionnalité peut être formaté et dans un temps raisonnable [Layaïda 97]. Nous avons pu le constater à travers la mise en œuvre de ce modèle (cf. section 5).

Nous verrons dans le chapitre 5 que nous avons utilisé des langages de présentation autres que le modèle Madeus 2.0 dans le but de générer des interfaces graphiques.

3.3. Bilan général

Dans cette section, nous avons identifié et défini les deux langages qui sont au cœur de notre système adaptable : le langage XSLT et le langage Madeus 2.0. Le langage XSLT est suffisamment expressif pour permettre l'écriture de transformations qui produisent des documents multimédias. De plus, comme ce langage est un standard, nous pouvons tirer profit des multiples outils associés à ce langage, en particulier les processeurs de transformation. Le second langage, Madeus 2.0, permet de décrire des présentations multimédias complexes. Il possède de bons atouts vis-à-vis de l'adaptation grâce à sa capacité de décrire des présentations flexibles. Enfin, l'architecture présentée dans la section 2.2 et les langages XSLT et Madeus 2.0 ont été décrits dans une publication présentée lors de la conférence internationale DDEP [Villard 00b]. De plus, nous avons effectué plusieurs expérimentations pour valider cette approche. Nous les décrivons dans la section 5.

4. Composants du système adaptable

L'objectif de cette section est dans un premier temps de décrire chaque composant de notre architecture (figure 35) en rappelant leur fonction principale, et ensuite d'identifier les techniques d'adaptation qui peuvent être appliquées dans chacun d'eux. Certaines de ces techniques ont été présentées dans le chapitre précédent. Au niveau de chaque composant, nous faisons apparaître les cas d'échec pouvant survenir suite à l'utilisation de ces techniques. Les langages employés par les composants sont ceux décrits dans la section précédente, c'est-à-dire XSLT et Madeus 2.0. Dans certain cas, nous serons amené à considérer d'autres langages lorsqu'il s'agit de montrer l'intérêt (ou non) de l'usage d'une technique donnée.

Étant donné que les composants sont organisés en chaîne, le traitement des cas d'échec ne peut se faire que par un composant situé avant celui qui a produit le cas d'échec. Aussi, nous commençons la description par le composant d'exécution, suivi du composant formatage puis du composant transformation. Nous verrons ensuite le rôle du superviseur d'adaptation avant de conclure.

4.1. Adaptation au niveau de l'exécution

4.1.1. Fonction du composant d'exécution

Le rôle du composant exécution est d'interpréter des informations de présentation encodées dans le document formaté et de les restituer sur le terminal de l'utilisateur (cf. figure 37). Ces informations sont encodées dans un format de bas niveau (le graphe d'exécution) pour permettre des traitements efficaces de la synchronisation [Sabry 99]. À partir de ces informations, le processus d'exécution assure les fonctions de présentation en tenant compte de l'interaction utilisateur. Il gère la synchronisation entre les médias, la navigation temporelle, ainsi que l'indéterminisme de la présentation. De plus, il se charge de gérer la création et la manipulation des médias du document.

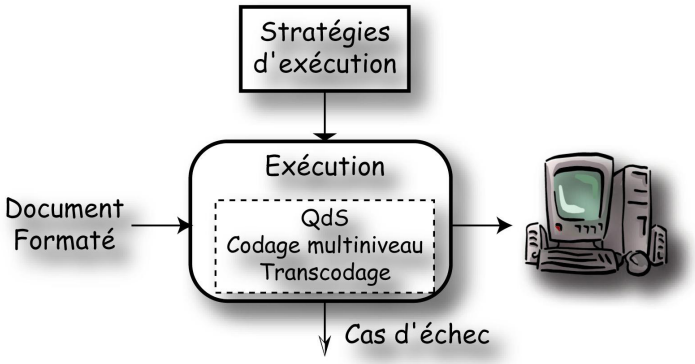


Figure 37. Le composant exécution.

Certaines des techniques d'adaptation pouvant être mis en œuvre au sein du processus d'exécution sont indépendantes du document formaté. Nous identifions ces techniques dans la sous-section suivante. Les techniques qui sont au contraire dépendantes du document formaté sont décrites dans la section 4.1.3. Nous décrivons ensuite des stratégies sur le choix de ces techniques et nous terminons par l'identification des cas d'échec.

4.1.2. Techniques d'adaptation indépendantes du document formaté

Le processus d'exécution est un processus de nature temps réel. Par conséquent, la qualité d'une présentation multimédia dépend fortement des ressources système et des ressources réseau disponibles. Pour limiter les effets de variabilité sur la disponibilité de ces ressources, les techniques de pré-chargement et de réservation de ressources sont mises en œuvre à ce niveau (cf. la section 3.1.1 du chapitre précédent).

De plus, comme le composant d'exécution gère l'accès et l'exécution des médias, c'est le seul composant à être capable de connaître la bande passante et la charge du CPU réellement consommée. Par conséquent, c'est seulement à ce niveau que peuvent être mis en œuvre des

mécanismes de réduction dynamique de bande passante et de charge CPU. Ces mécanismes reposent sur l'encodage multiniveau de médias ou de façon plus fine sur la distillation de médias. Les techniques de transcodage qui conservent la composition temporelle et spatiale, comme la conversion d'une vidéo en une suite d'images, ou le changement de format de GIF vers JPEG peuvent aussi être utilisées au niveau de l'exécution.

4.1.3. Techniques d'adaptation dépendantes du document formaté

L'application des techniques d'adaptation ci-dessus dépend des contraintes imposées par le système et le réseau. Une deuxième catégorie de techniques applicables est relative aux contraintes imposées par l'auteur à travers la spécification du scénario. Les techniques pouvant être appliquées dépendent alors des informations transmises au niveau du processus d'exécution. Lorsque la durée des médias permet un certain degré de flexibilité, le fait de conserver l'intervalle de valeur pendant la phase d'exécution permet de retarder l'exécution des médias [Sabry 99]. Cela permet, entre autres, d'attendre les données nécessaires et/ou les ressources CPU suffisantes pour jouer le média tout en conservant la cohérence temporelle du document.

4.1.4. Stratégies d'exécution

Comme nous l'avons vu dans le chapitre 3, le même résultat d'adaptation peut être obtenu en utilisant des techniques différentes. Par exemple, le saut d'images et la distillation permettent tous deux de réduire la bande passante consommée. La différence se situe au niveau de la perception du document par l'utilisateur. Par conséquent, plusieurs *stratégies* pour choisir la technique d'adaptation appropriée peuvent être appliquées. Ces stratégies peuvent être définies indépendamment du document de présentation. Elles peuvent être contrôlées par le superviseur d'adaptation ou plus simplement par l'utilisateur à travers un ensemble de préférences. Le choix d'une stratégie peut également dépendre du contexte applicatif, par exemple pour le document de type SlideShow, une bonne stratégie d'adaptation pour réduire la bande passante est de diminuer la qualité des images de la vidéo plutôt que la qualité du son.

4.1.5. Cas d'échec

Dans le cadre où les techniques de réservation de ressources ne sont pas mises en œuvre, ou parce que les ressources disponibles ne sont de toutes façons pas suffisantes, le composant exécution peut arriver dans un état d'échec. Le processus d'exécution entre dans un tel état lorsque les techniques d'adaptation applicables à ce niveau ne suffisent plus à compenser la diminution des ressources. Dans ce cas, il est nécessaire d'appliquer des modifications plus importantes sur l'organisation du document, en commençant par le niveau du formatage.

4.2. Adaptation au niveau du formatage

4.2.1. Fonction du composant formatage

Le formatage est l'opération qui consiste à interpréter les informations contenues dans un document de présentation afin de produire des informations directement interprétables par le processus d'exécution (cf. figure 38). Ces informations sont représentées par des structures internes de l'application. Dans le cas de Madeus 2.0, le formatage résout le système de

contraintes induit par la spécification des relations (temporelles, spatiales et hiérarchiques). Il calcule ainsi le positionnement temporel et spatial absolu de chaque acteur de la présentation.

Comparé au processus d'exécution, le formatage a une connaissance sémantique de la présentation. Cette sémantique porte notamment sur l'organisation des médias à travers les dimensions multimédias. Le degré de sémantique dépend du modèle de document de présentation à formater. Plus ce degré sémantique est élevé, et plus des techniques d'adaptation efficaces pourront être mises en œuvre. Par exemple, pour les modèles de présentation spécifiant uniquement des placements absolus, la technique de filtrage ne peut pas être appliquée de façon satisfaisante. Le média est supprimé de la présentation, mais comme les autres médias sont positionnés de façon absolue, l'espace temporel et spatial rendu disponible ne peut pas être utilisé pour optimiser le positionnement de ces autres médias. Ce n'est pas le cas lorsque les placements sont relatifs.

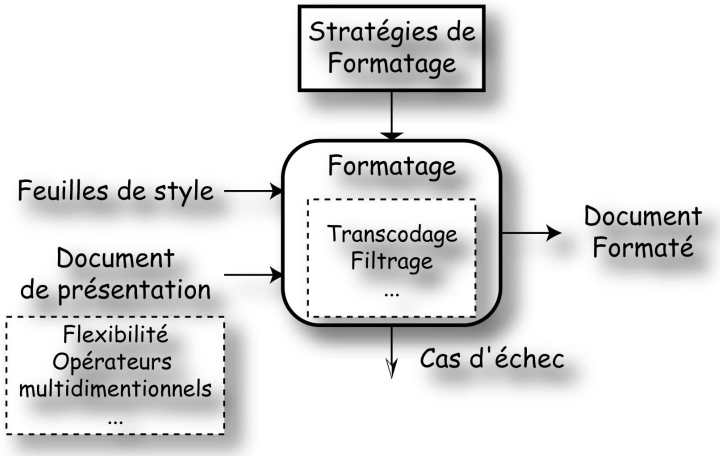


Figure 38. Le composant formatage.

Nous étudions ci-dessous les différentes techniques d'adaptation en montrant l'impact sur le modèle du document de présentation et sur le résultat de l'adaptation. Nous montrons ensuite quels peuvent être les cas d'échec et les stratégies pour formater le document.

4.2.2. Techniques d'adaptation

4.2.2.1 Transcodage

Les techniques de transcodage peuvent être utilisées dans le composant d'exécution mais aussi dans les autres composants notamment ceux pour lesquels la remise en cause des structures de présentation spatiales et temporelles est possible. Par exemple, le processus de formatage peut décider de transcoder un objet de type texte en un audio. En effet, le formatage est capable de déterminer si aucun objet de type audio n'entre en conflit avec l'objet audio résultat du transcodage. L'objet texte initial est remplacé par un objet de taille nulle ce qui permet de conserver les relations spatiales associées. Les relations temporelles sont modifiées en remplaçant l'ancien objet texte par le nouvel objet audio. Une des conditions pour appliquer ce type de transcodage est que la durée de l'objet audio, calculée par le processus de formatage, doit être la même que la durée de l'objet texte spécifiée par l'auteur. Dans le cas d'une spécification non flexible, les cas de correspondance sont rares. Ce type de transcodage

n'est donc réellement exploitable qu'à la condition d'avoir une présentation flexible. De façon générale, tous les types de transcodage (conversion, distillation, etc.) présentés dans la section 3.1.3 du chapitre précédent peuvent être employés par le processus de formatage. La qualité du résultat dépend de la performance du transcodeur.

4.2.2.2 Filtrage

Le processus de formatage est capable de filtrer un média et de conserver une mise en page cohérente. Cependant, la décision de filtrer ou non ne peut pas se faire à ce niveau car elle requiert une connaissance sémantique du média et du document (cf. chapitre 3 section 3.1.4). Une solution est d'associer à chaque média un ensemble de méta-données pour indiquer notamment sa catégorie. À partir de cette information, le processus de formatage peut décider automatiquement de filtrer, par exemple, les images publicitaires.

4.2.2.3 Flexibilité

Les informations de présentation contenues dans le document peuvent être spécifiées de façon à offrir plus ou moins de latitude sur le calcul des valeurs effectives lors de la présentation. Par exemple, si un objet A est positionné aux coordonnées (10, 10) et qu'un objet B est lié à l'objet A par la relation « à gauche de », les coordonnées de B sont situées dans une boîte de coordonnées (10, -∞)-(+∞, +∞). Le rôle du formatage est de calculer pour chaque objet des coordonnées précises qui sont cohérentes avec le restant des contraintes du document. Cette flexibilité peut s'appliquer sur les caractéristiques d'un média et aussi sur l'organisation de ceux-ci. Dans Madeus 2.0, les deux besoins sont déjà supportés, en spécifiant des durées flexibles et des relations flexibles.

4.2.2.4 Intégration forte des dimensions multimédias

Jusqu'à présent, les modèles de documents existants n'exploitent pas réellement l'intégration de plusieurs dimensions dans un même document. Hormis les animations qui mettent en jeu à la fois la dimension temporelle et spatiale, aucune autre forme d'intégration n'est recensée. Or l'intégration forte des dimensions multimédias est un moyen qui peut permettre d'optimiser le placement des médias. Par exemple, une façon d'organiser les médias est de les positionner les uns en dessous des autres dans une région. Lorsque la limite inférieure de l'espace d'affichage est atteinte, une nouvelle région est créée et jouée temporellement après la première région. Un autre exemple est une extension de l'élision syntaxique présentée par Bickmore (cf. chapitre 3 section 3.1.6). Le début d'une phrase peut être montré graphiquement et lorsque l'utilisateur l'active, le reste de la phrase est montrée en utilisant un déplacement progressif.

4.2.2.5 Organisation sémantique des acteurs

Alors que les deux techniques précédentes proposent des extensions du modèle au niveau des dimensions temporelle et spatiale, une autre technique est d'étendre la structure logique du modèle de présentation, c'est-à-dire la structure d'acteurs pour Madeus 2.0. Ces extensions, qui n'interfèrent pas directement avec le scénario multimédia, peuvent servir pour appliquer des transformations prédéfinies au sein même du formatage. Les extensions regroupent les concepts de présentations les plus répandus et communs à de nombreux domaines. C'est le cas par exemple de la table des matières et de la barre de navigation. Avec les techniques actuelles, lorsqu'un auteur souhaite afficher une table des matières, il doit écrire une feuille de

transformation. Cette feuille de transformation, et de façon générale la majorité des feuilles de transformation générant des tables des matières, identifie les *entrées* de la table et décrit la présentation de ces entrées. Seule la première étape dépend du document source, alors que la seconde étape peut être réutilisée pour d'autres documents source. En définissant un vocabulaire précis pour décrire les tables des matières au sein du modèle de présentation, les présentations peuvent facilement être réutilisées. De plus, il est possible d'offrir plusieurs présentations répondant aux contraintes et aux préférences relatives au contexte utilisateur.

4.2.2.6 Abstraction des périphériques

Puisque plusieurs terminaux peuvent être utilisés pour accéder à une présentation, plusieurs modalités d'entrée peuvent être utilisées pour interagir. Au lieu de spécifier chacune de ces interactions, nous avons choisi d'abstraire ces modalités d'entrée. La majorité des terminaux possèdent un périphérique de pointage, comme la souris pour les stations de travail, le stylet pour les PDA, un écran tactile pour les kiosks interactifs. Nous introduisons un nouveau module dans le modèle de présentation, nommé *modalité d'entrée*, qui définit un ensemble de périphériques abstraits. En particulier, ce module contient le périphérique pointerCursor. À ce périphérique sont associés plusieurs événements, entre autres, le déplacement (onMove), l'activation (onActivation), la double activation (onDoubleActivation), l'entrée dans un objet (onEnter) et la sortie d'un objet (onExit).

Par exemple, le changement de couleur d'une entrée de la table des matières lorsque l'utilisateur passe au dessus de l'entrée est spécifié comme suit :

```
1.  …
2.  <actor>
3.      <text id="slide1_toc" style="foreground:black">
4.          <animateMotion id="colorAnim" attributeName="foreground"
5.                              values="from:black to:white"/>
6.      </text>
7.  </actor>
8.  <temporal>
9.      <interval actor="slide1_toc">
10.         <interval hrefActivation="pointerCursor.onEnter" actor="slide1_toc.colorAnim"/>
11.     </interval>
12. </temporal>
13. …
```

Tous les éléments de cet exemple ont été décrits dans la section 3.2, hormis l'élément animateMotion qui fait partie du module animation. Le lecteur peut se référer à la norme SMIL 2.0 pour la description de ce module.

Le module modalité d'entrée permet de définir un document qui est portable sur plusieurs terminaux. Il permet ainsi de faciliter l'adaptation sans toutefois répondre complètement au problème. En effet, le document précédent ne peut pas se jouer sur un téléphone cellulaire qui n'a pas de modalité de pointage.

4.2.3. Stratégies de formatage

Enfin, pour les mêmes raisons évoquées dans la section précédente, plusieurs stratégies peuvent être élaborées pour formater le document de présentation. Chacune de ces stratégies

définit des règles qui permettent, en particulier, de choisir la feuille de style à appliquer ou les types transcodage à privilégier.

Comparé à la transformation, le formatage est conçu pour un langage (de présentation) déterminé. Lorsqu'il s'agit de faire de l'adaptation qui dépend de la sémantique du domaine, le formatage n'est pas suffisant. C'est pourquoi il est nécessaire d'ajouter un composant de transformation qui agit à un niveau d'abstraction plus élevé du document.

4.2.4. Cas d'échec

Comme l'adaptation est réalisée par le processus de formatage, les capacités d'adaptation et, par conséquent, les cas d'échec de ce composant dépendent du modèle de document de présentation. Pour certains des modèles existants (HTML, SVG, etc.), le formatage ne produit aucun cas d'échec, à condition que le document à formater soit valide. Ce n'est cependant pas le cas notamment des documents SMIL ou Madeus qui, bien qu'ils soient valides syntaxiquement, peuvent être *incohérents*. Dans ce cas, le processus de formatage peut produire des cas d'échec.

Deux niveaux de cas d'échec sont identifiés selon l'origine de la contrainte :

- Au niveau du contexte utilisateur : les cas d'échec peuvent être liés à la non satisfaction des contraintes imposées par l'utilisateur à travers son profil. La détection de ce type d'échec repose sur la confrontation du résultat de l'adaptation avec les paramètres du contexte utilisateur. Il faut noter que cette détection doit se faire aussi pour le processus de transformation (cf. section suivante) ;

- Au niveau du modèle : ces cas d'échec sont dus principalement aux incohérences temporelles qui peuvent être de nature qualitative, quantitative ou encore indéterministe [Layaïda 96]. Par exemple, une incohérence quantitative résulte de l'impossibilité de trouver une solution de placement et de dimensionnement des objets à l'intérieur d'une boîte englobante dont la taille a été fixée par l'auteur.

Dans les deux cas, une solution est de produire un formatage approximant la solution réelle. Supposons, par exemple, que l'auteur veuille une durée de présentation maximum de 20 secondes. Supposons de plus que deux objets A et B sont liés par une relation temporelle de type séquence et que la durée de ces objets est de 11s, et que le formatage affecte la valeur 22s comme instant de fin de B. Cela transgresse la contrainte imposée par l'auteur. Un formatage approximatif consiste, par exemple, à relâcher la relation temporelle en une relation de type « overlaps », pour ajuster l'instant de fin à 20 secondes. Cette technique a été expérimentée en s'appuyant sur la recherche de relations voisines [Diaz 01].

4.3. Adaptation au niveau de la transformation

4.3.1. Fonction du composant de transformation

La transformation est le processus qui permet de convertir un document source vers un document cible en suivant des règles spécifiées dans des feuilles de transformation (cf. figure 39). Ces feuilles de transformation peuvent être paramétrées. Par exemple, un des paramètres de la feuille de transformation SlideShow est le choix de montrer ou non la table des matières (cf. figure 33).

À la différence du formatage, la transformation a une connaissance du domaine représenté par le document source. Par conséquent, c'est au niveau de la transformation que des adaptations liées au domaine du document peuvent être effectuées. De plus, contrairement au formatage, la transformation est un processus générique : le modèle du document source n'est pas *a priori* connu. Comme les méta-modèles existants ne permettent pas de formaliser la sémantique d'un modèle (cf. chapitre 2 section 2.3), l'interprétation de celui-ci ne peut se faire que par un auteur à travers la spécification des transformations.

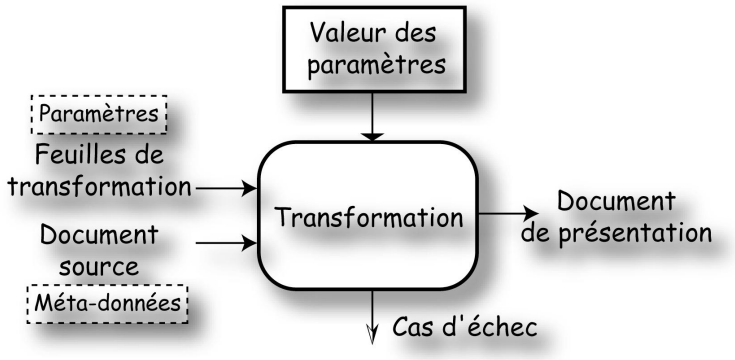


Figure 39. Le composant transformation.

Par rapport aux processus précédents, aucune technique d'adaptation n'est ajoutée dans le processus de transformation. C'est au niveau de la spécification et de la sélection de la transformation que s'effectue l'adaptation. Par exemple, pour la présentation SlideShow sur un PDA en mode différé, le transcodage de la vidéo en son (qui consiste en un simple filtrage des images de la vidéo) s'effectue en transformant syntaxiquement l'objet vidéo du document source vers un objet de type son. Comme c'est un auteur qui spécifie les transformations, il est nécessaire d'étudier les méthodes qui permettent de minimiser l'effort de conception. Pour le langage XSLT, nous en avons identifié trois : la réutilisation de transformation, le paramétrage des transformations et l'ajout de méta-données sur le document source. Nous étudions ces trois méthodes ci-dessous. Nous verrons ensuite les cas d'échec du processus de transformation et les stratégies sur le choix des transformations.

4.3.1.1 Méthodes de conception

4.3.1.2 Réutilisation

Dans XSLT, il existe deux façons de réutiliser des transformations : la réutilisation de modules, et la réutilisation de type pipeline. Ces deux types de réutilisation sont des concepts bien connus en génie logiciel [Rumbaugh 97]. Dans le cadre du premier type de réutilisation, les modules peuvent être assemblés de deux façons : par inclusion et par importation. La distinction entre ces deux façons repose sur la notion de niveau de profondeur d'une règle. Dans le premier cas, les règles incluses sont au même niveau, et par conséquent elles sont traitées de la même façon que les règles contenues dans le module initial. Dans le second cas, les règles sont incluses avec un niveau de profondeur inférieur par rapport aux règles du module initial qui peuvent alors surcharger les règles incluses. Ces deux solutions d'assemblage permettent de réutiliser des fragments de transformation.

La réutilisation de type pipeline consiste à chaîner deux ou plusieurs processus de transformation. Elle permet de réutiliser une transformation dans sa globalité.

La réutilisation de modules permet d'avoir une approche de conception par généralisation et par spécialisation [Rumbaugh 97]. Dans le contexte de l'adaptation, la première approche consiste à écrire des modules pour des terminaux ayant peu de fonctionnalité (Wap), et à compléter ces modules par de nouvelles. La seconde approche, au contraire, repose sur une spécification pour des terminaux avec beaucoup de fonctionnalités et de spécialiser ces modules en restreignant les fonctionnalités. L'utilisation de l'une ou l'autre méthode dépend de l'expérience acquise durant la conception de documents adaptables. Nous verrons quelques pistes dans la section 5 à travers plusieurs expérimentations.

4.3.1.3 Paramétrage

L'exécution des feuilles de transformation peut dépendre d'un ensemble de paramètres dont la valeur est donnée au début du processus de transformation. Ces paramètres sont utilisés comme donnée copiée dans le document cible, et/ou par les instructions du langage pour effectuer des choix, des filtres, etc. L'usage de ces paramètres dans le contexte de l'adaptation permet de prendre en compte un paramètre d'adaptation au sein même de la transformation et de réaliser ainsi une adaptation plus fine. Par exemple, un des paramètres de la feuille de transformation SlideShow est la taille de la police de caractères pour afficher les items de liste. Ce paramètre est utilisé sur la décision de l'auteur au sein de plusieurs règles de transformation.

4.3.1.4 Méta-données

Le document source peut contenir des méta-données sur lesquelles des transformations peuvent s'appuyer. L'intérêt est de pouvoir réutiliser ces feuilles de transformations sur plusieurs modèles de document différents mais partageant un même ensemble de méta-données. Cette approche est similaire à celle étudiée dans la section 4.2.2.2 concernant l'ajout de méta-données pour permettre le filtrage de média ou de fragment de document. Cependant, l'avantage des méta-données au niveau de la transformation est d'une part une plus grande souplesse d'extension du modèle de document par de nouvelles méta-données, grâce à la facilité d'écriture de feuilles de transformation par rapport au codage d'un processus de formatage. Et d'autre part, l'usage de ces méta-données n'est pas restreint aux traitements prévus par le formatage (comme le filtrage).

4.3.2. Cas d'échec

Le seul cas d'échec de la transformation correspond à un document cible qui ne respecte pas le modèle lui étant associé. Il est en effet impossible de garantir de façon statique, c'est-à-dire avant l'exécution d'une transformation, que quelque soit le document source en entrée de la transformation, le document cible respecte le modèle de présentation. Des travaux sont en cours pour tenter de trouver des solutions à ce problème difficile [Audebaud 00, Milo 00, Hosoya 00, Tozawa 01].

4.3.3. Stratégies

Plusieurs feuilles de transformation peuvent être définies pour un même modèle de document. Il est donc nécessaire de mettre en place un mécanisme pour choisir quelle feuille utiliser pour

un contexte utilisateur donné. De plus, ce mécanisme doit être en mesure d'affecter une valeur aux paramètres de la feuille de transformation choisie. Dans certains cas, il doit aussi être capable d'assembler dynamiquement plusieurs modules de transformation et de chaîner les transformations. Par exemple, pour adapter la présentation SlideShow (cf. figure 33), produite par la feuille de transformation appelée Show (cf. section 5.1.1), sur un téléphone cellulaire Wap, une solution d'assemblage est la suivante :

- paramétrer la feuille de transformation Show de telle façon à minimiser l'espace nécessaire à l'affichage, notamment en supprimant la table des matières ;
- choisir les modules de transformation proches des fonctionnalités du langage de destination WML ;
- chaîner le document de présentation Madeus 2.0, produit par l'application de la feuille de transformation Show, avec une feuille de transformation qui le transforme en WML.

4.4. Rôle du superviseur d'adaptation

Le rôle du superviseur d'adaptation est de contrôler la communication entre les différents composants d'adaptation. Lorsqu'un composant entre dans un état d'échec, le superviseur en est informé. Il agit alors sur les stratégies disponibles à chaque niveau de l'architecture d'adaptation pour remédier à cette situation d'échec.

La fonction du superviseur est aussi de confronter le contexte utilisateur avec les documents multimédias et les feuilles de transformations existantes. Le superviseur agit alors sur les stratégies disponibles sur chaque composant et sur l'assemblage des composants de transformation et aussi de formatage. Nous verrons des exemples d'assemblage manuel dans la partie expérimentation. De plus, nous considérons que c'est l'utilisateur qui choisit la ou les feuilles de transformation qui correspondent à son profil.

4.5. Bilan

Dans cette section, nous avons décrit la fonction de chacun des composants du système adaptable et nous avons identifié les techniques d'adaptation utilisées par ces composants. Nous avons défini la fonction générale de chaque composant, le contenu de ces composant et l'interaction entre ces composants. Nous avons aussi montré qu'il existait des cas d'échec plus ou moins détectables et qu'il était possible de mettre en place plusieurs stratégies d'adaptation au sein de chaque composant. Ce système constitue une base fondamentale à partir de laquelle de nombreux problèmes restent à résoudre.

Les paramètres d'adaptation pris en compte par ces composants sont plus ou moins identifiés selon le type du composant. En ce qui concerne le processus d'exécution, de par sa nature temps-réel (débit réseau, charge CPU), il permet de traiter les paramètres temps réels identifiés dans la section 2.3 du chapitre précédent. Le formatage est un processus pour lequel il est possible d'implanter des algorithmes performants et spécifiques au modèle de présentation. De plus, dans le cas idéal ce processus connaît les réelles intentions de l'auteur vis-à-vis du scénario multimédia, à condition que ces intentions puissent être formalisées par le modèle de présentation. Par conséquent, ce composant permet donc de traiter les paramètres dynamiques (taille de la fenêtre, préférences de l'utilisateur, etc.) et de présentation. Enfin, les paramètres restants (relatifs au domaine) sont pris en compte par le processus de transformation.

Cette vision schématique n'est cependant pas aussi nette dans la réalité. La frontière entre la transformation et le formatage varie selon l'expressivité du modèle de présentation. Par exemple, lorsque le modèle de présentation n'offre aucune flexibilité sur la taille des caractères, c'est pendant la transformation que les calculs doivent être effectués par l'auteur. De façon générale, le surcoût de conception peut être diminué grâce à un modèle de présentation suffisamment expressif. Le problème est d'identifier les fonctionnalités pertinentes au niveau de la présentation qui permettent de faciliter l'adaptation. La description des tables des matières dans le modèle de présentation (cf. section 4.2.2.5) fait sûrement partie de ces fonctionnalités à rajouter.

Concernant le besoin de présenter les documents multimédias appartenant à une classe, cette architecture représente déjà une réponse pertinente. La classe de document multimédia est définie à la fois par le modèle de document source et par les feuilles de transformation.

Dans la suite de ce chapitre nous relatons les différentes expériences que nous avons menées sur ce système adaptable.

5. Expérimentations

Afin de valider certains des concepts que nous avons présentés dans les sections précédentes, plusieurs expérimentations ont été effectuées. Le cœur de chacune de ces expérimentations est la technique de transformation pour laquelle nous avons défini de nombreuses spécifications.

Nous décrivons dans la section 5.1 les expérimentations que nous avons effectuées sur des modèles de documents métier particuliers. Les secondes expérimentations traitent du problème de la transformation entre profils langagiers existants. Nous définissons ce problème et relatons les expériences associées dans la section 5.2.

5.1. Adaptation de documents multimédias appartenant à une classe

Notre système de production de présentations multimédias adaptées a été expérimenté à travers plusieurs modèles de document métier. Dans la suite de cette section nous décrivons deux de ces expérimentations, la première sur le modèle de document SlideShow et la seconde sur le modèle de document ATA. Nous effectuons ensuite un bilan de ces expérimentations.

5.1.1. Le modèle SlideShow

La première expérimentation que nous avons effectuée porte sur les documents de type SlideShow dont nous avons donné des exemples tout au long de ce chapitre.

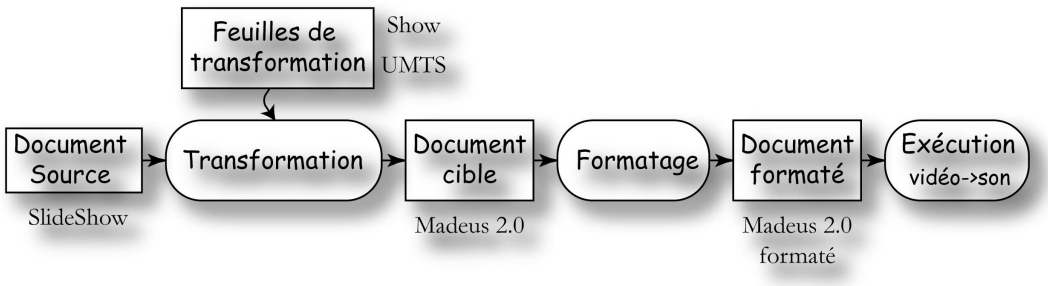


Figure 40. Processus de production d'une présentation de type SlideShow.

Le processus du système de présentation pour ce modèle de document est présenté figure 40. Le document source est de type SlideShow. Le résultat de la transformation produit un document de type Madeus 2.0.

Plusieurs feuilles de transformation ont été écrites pour prendre en compte les paramètres d'adaptation suivants : le choix d'un style de présentation, le choix entre une présentation interactive ou non, le choix de montrer ou non la table des matières et le choix entre une présentation en directe ou différée. Le rôle de ces paramètres a été détaillé dans la section 2.1.

À partir de ces paramètres, les caractéristiques suivantes sont déduites :

- la taille des polices de caractères de chacun des éléments du transparent, calculée proportionnellement à la hauteur de l'écran ;
- le choix de montrer ou non la table des matières. Ce choix est une fonction de la taille de l'écran et du type de présentation (en direct ou différée). La politique mise en place est de laisser au moins un espace d'une taille de 640x480 pixels pour montrer la présentation. Si l'espace disponible est supérieur, la table des matières est montrée dans le cas où la présentation est différée ;
- le choix ou non de montrer la vidéo de la présentation. Ce choix dépend de la taille de l'écran et du choix de la présentation différée ou live et du choix de montrer la table des matières. Lorsque celle-ci n'est pas montrée et que la présentation est différée, alors seul le son de la vidéo est diffusé.

Figure 41. Présentation du document SlideShow section 2.1 adaptée à un téléphone UMTS.

Grâce à ces paramètres en entrée de la feuille de transformation Show (cf. figure 42), plusieurs présentations peuvent être générées, comme celle figure 33 de la section 2.1. Notamment, la présentation ci-dessous figure 41, issue des mêmes feuilles de transformation, est le résultat de la génération du même document, de la section 2.1, lorsque la taille de l'écran est inférieure à 640x480. La table des matières n'est pas affichée et la présentation est dans le mode différé : la vidéo n'est pas montrée, seul le son est diffusé. Cette présentation est donc particulièrement adaptée à un téléphone UMTS[16] de troisième génération sur lequel est installé un « player » supportant le langage SMIL Basic.

La spécification des feuilles de transformation est organisée selon l'architecture illustrée figure 42. Sept feuilles de transformation ont été conçues : trois pour la partie table des matières (tdmActeur, tdmSpatial et tdmTemporel), trois pour la partie présentation des transparents (showActeur, showSpatial et showTemporel) et une feuille de transformation qui agrège l'ensemble de ces feuilles (Show). Les liens temporels entre la table des matières et la présentation sont décrits dans le module tdmActeur.

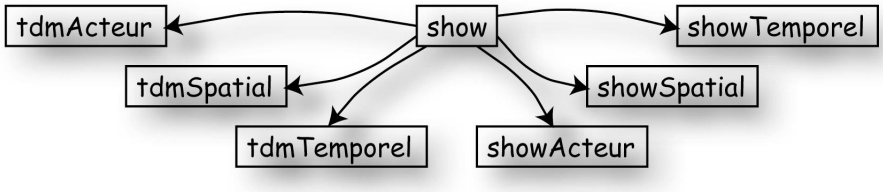


Figure 42. Architecture des feuilles de transformation pour le modèle SlideShow.

Les caractéristiques énoncées ci-dessus sont implantées par plusieurs paramètres globaux déclarés dans la transformation Show. Selon la valeur de ces paramètres, la présentation diffère et par conséquent les fonctionnalités produites par les transformations ne sont pas les mêmes. Cette relative flexibilité permet d'augmenter les possibilités d'adaptation à différents terminaux.

Les feuilles de transformation ont été écrites dans le but de générer des documents de type Madeus 2.0. Afin de pouvoir visualiser ces présentations en dehors du système de présentation Madeus, nous avons conçu deux transformations, l'une vers le profil SMIL 2.0 et la seconde vers le profil SMIL Basic (cf. sections 5.2.3 et 5.2.4).

5.1.2. Le modèle ATA

Une seconde expérimentation a été effectuée sur les documents aéronautiques de type ATA 2100 (cf. section 2.2.2.2 du chapitre 2). Ces documents décrivent une tâche de maintenance d'un avion. Cette expérimentation a été effectuée dans le cadre de la collaboration entre l'Aerospatial Airbus Matra et l'INRIA. L'objectif principal de cette collaboration est de proposer une méthode pour présenter des documents aéronautiques multimédias. Cette collaboration a fait l'objet de deux rapports d'avancement [Villard 00a, Villard 01c] dans lesquels les différents modèles et feuilles de transformation sont décrits de façon détaillée.

[16] Universal Mobile Telecommunications System, http://www.umts-forum.org/.

La figure 43 montre une présentation possible d'une tâche de maintenance. Le scénario de ce document est le suivant. Lorsque le lecteur a choisi une tâche à afficher en utilisant la table des matières illustrée à gauche, la présentation multimédia de cette tâche de maintenance suit le scénario suivant :

- les mises en garde (représentées par les éléments warnings) sont d'abord présentées en séquence de façon sonore et visuelle ;
- puis les informations et les tâches *job-setup* sont présentées en même temps et dans une même fenêtre. Un bouton procédure est affiché afin de poursuivre la présentation ;
- lorsque le lecteur appuie sur le bouton procédure, la vidéo associée à la tâche et le texte des différentes étapes de la tâche sont visualisés ;
- au fur et à mesure de la progression de la vidéo, le texte correspondant à l'image présentée dans la vidéo est mis en surbrillance. À tout moment, le lecteur a la possibilité de suspendre le déroulement de la vidéo et de la recommencer depuis le début ;
- à la fin de la présentation de la vidéo, les informations de *close-up* sont présentées en même temps et dans une même fenêtre.

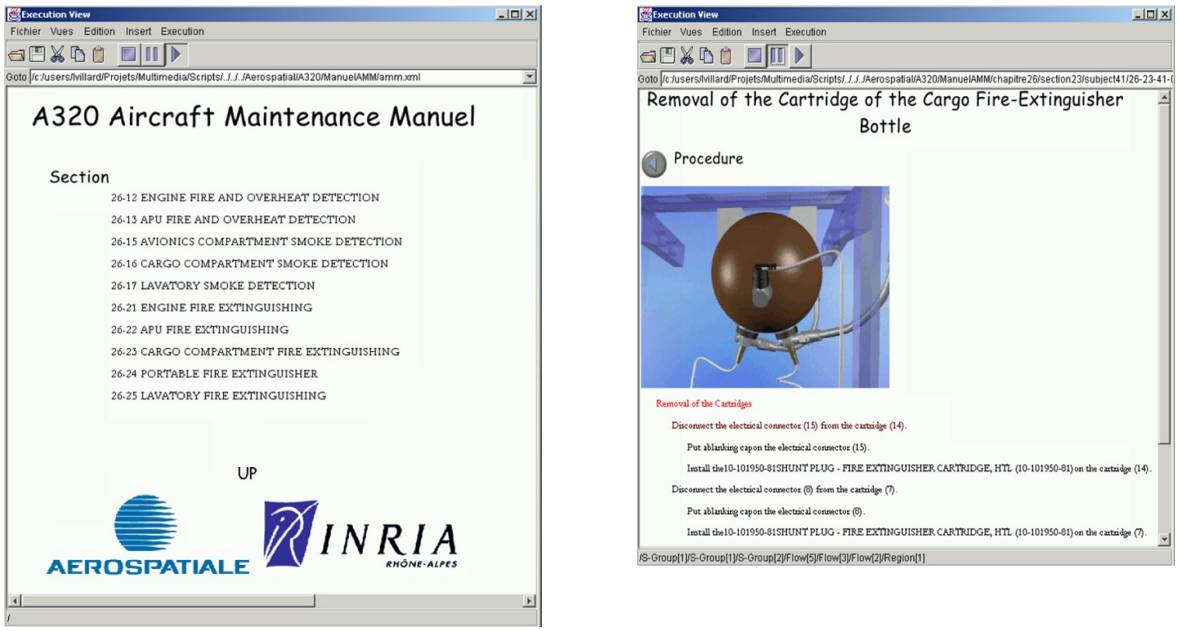


Figure 43. Présentation de la tâche de démontage de la cartouche de l'extincteur d'un cargo.

Les choix de conception pour le modèle SlideShow ont été repris pour le modèle ATA. Trois jeux de feuilles de transformation ont été réalisés pour : la navigation temporelle, la table des matières hiérarchique, et la présentation d'une tâche de maintenance. Chacun de ces jeux comporte quatre fichiers correspondant au trois dimensions du modèle Madeus (acteur, temps, et spatial) et au fichier racine agrégeant ces trois dimensions.

Cette expérimentation a été effectuée pour montrer l'adéquation entre le langage XSLT et Madeus 2.0 pour présenter des documents issues d'une même classe (ici l'ATA 2100). Aucun

paramètre d'adaptation, autres que ceux pris en compte par le système adaptable, n'a été ajouté.

5.1.3. Bilan

Ces deux expérimentations ont été implantées dans le prototype de présentation et d'édition Madeus développé au sein du projet Opéra. Un processeur de transformation XSLT a été greffé entre l'analyseur de document XML et l'interpréteur des documents Madeus 2.0. Cet interpréteur a été modifié pour prendre en compte les fonctionnalités du langage Madeus décrites dans la section 3.2. Plus précisément, nous avons pris en compte le découpage d'une spécification Madeus 2.0 en trois structures distinctes (acteur, temporel et spatial), nous avons étendu le modèle temporel par les opérateurs **seq** et **excl**, et nous avons ajouté plusieurs opérateurs spatiaux. Chacune de ces modifications ont porté sur le module de formatage et sur le module d'exécution. En conclusion, ce prototype est pleinement opérationnel et permet d'exécuter en un temps raisonnable l'ensemble des documents décrits dans cette section.

5.2. Transformations entre profils

Dans cette section nous décrivons les expérimentations que nous avons effectuées sur la conversion entre documents appartenant au même langage mais définis en utilisant des profils langagiers différents.

5.2.1. Principes

Le principe de la transformation entre profils langagiers est illustré figure 44 [Lemlouma 01]. Le module A est transformé vers deux modules F et C et le module G n'a pas d'équivalent dans le second profil. Il convient dans ce cas d'expliciter les conséquences sur la présentation. Les modules E et D appartiennent aux deux profils et donc le document ne nécessite pas, pour les éléments appartenant à ces modules, de transformation particulière.

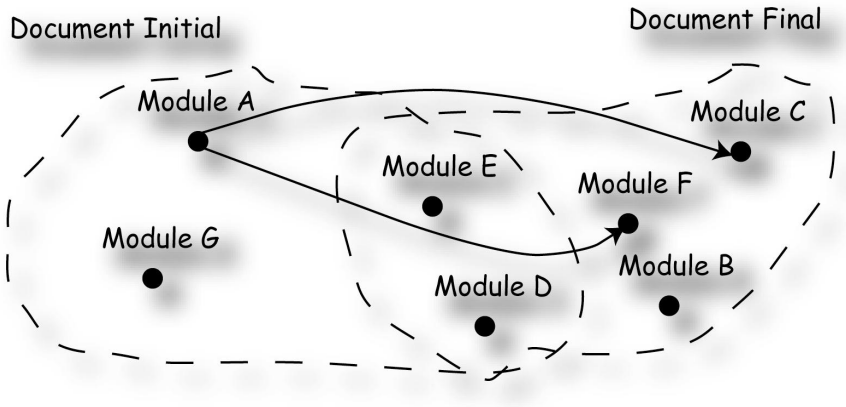


Figure 44. Principe de la transformation entre profils langagiers.

Nous avons appliqué ce principe au standard SMIL 2.0 et au langage Madeus 2.0. Trois types de conversion sont décrits : du profil SMIL2.0 vers le profil xhtml+smil [XHTML+SMIL 01], du langage Madeus 2.0 vers le profil SMIL 2.0 et du langage Madeus 2.0 vers le profil SMIL Basic. Nous avons choisi de réaliser la première conversion car xhtml+smil a été quelque temps le seul profil SMIL 2.0 implanté. La seconde conversion a été réalisée dans le but de

visualiser des documents Madeus 2.0 sur des « players » largement diffusés. Enfin la troisième conversion permet de montrer des documents Madeus 2.0 sur la seule implantation d'un « player » SMIL Basic sur les assistants personnels de type PocketPC (qui est réalisé au sein du projet Opéra). Pour chacune de ces conversions nous décrivons uniquement les cas de transformations complexes, c'est-à-dire autres que l'identité, le changement de type ou de nom d'attribut. Nous indiquons également les fonctionnalités sui sont perdues lors de la transformation.

5.2.2. Du profil SMIL 2.0 vers le profil xhtml+smil

Le profil xhtml+smil [XHTML+SMIL 01] est basé sur les modules du langage XHTML et le module d'annotation Ruby [Ruby 01]. De plus, des fonctionnalités temporelles sont ajoutées en intégrant quelques modules de SMIL 2.0. L'objectif de cette section est de montrer qu'un document écrit dans le profil SMIL 2.0 peut être converti simplement vers un document xhtml+smil grâce à une transformation de type XSLT.

Pour les modules communs aux deux profils, la transformation identité est appliquée. Pour les autres modules, nous effectuons une transformation particulière. Nous décrivons ci-dessous les transformations non triviales :

- Module MinMaxTiming : les attributs `min` et `max` fournissent à l'auteur un moyen pour contrôler les bornes minimum et maximum de la durée active d'un média. Pour prendre en compte ces attributs, il est nécessaire de calculer la durée active d'un média. Or il arrive que ce calcul puisse se faire uniquement pendant la présentation (suite à un événement utilisateur). Par souci de simplification, ces attributs sont remplacés par l'attribut `dur` en respectant les règles suivantes :
    - si un attribut `dur` existe déjà, les attributs `min` et `max` sont filtrés ;
    - si aucun attribut `dur` n'existe, alors il est ajouté avec comme valeur la moyenne entre l'attribut `min` et l'attribut `max`. Si un des deux attributs n'est pas spécifié, alors la valeur de l'attribut `dur` correspond à la valeur de l'attribut spécifié ;
- Module HierarchicalLayout : ce module permet d'imbriquer des régions spatiales. En CSS, cette fonctionnalité est aussi offerte. Cependant, à la différence de SMIL2.0, en xhtml+smil les structures temporelles et spatiales sont identiques. Deux solutions sont possibles : soit la structure spatiale est recopiée complètement, soit le convertisseur calcule la position absolue du média. Le choix entre les deux n'est pas évident, car chacune a des avantages et inconvénients. La première solution produit un document cible plus gros mais la transformation est plus simple et ne nécessite pas de connaître les unités possibles pour spécifier les dimensions. À l'opposé, la seconde solution produit un document plus concis mais nécessite de connaître les unités et la transformation est plus complexe. Les deux solutions consomment des ressources différentes, de la bande passante pour la première et des ressources CPU pour la seconde. Nous avons expérimenté la seconde pour montrer l'expressivité du langage XSLT.

Plusieurs modules sont transformés en scripts équivalents en utilisant le langage JavaScript, car aucune équivalence n'existe en utilisant uniquement les éléments du profil. C'est le cas en particulier du module AccessKeyTiming et MultiWindowsLayout. De plus, pour prendre en compte le module MediaMarker, il est nécessaire d'étendre le langage XSLT (ce qui est permis

par la norme) par des fonctions supplémentaires permettant d'accéder aux marqueurs contenus dans un média.

Le seul module n'ayant pas d'équivalent dans xhtml+smil est le module PrefetchControl. Ce module permet de précharger un média en vue de son utilisation ultérieure. Par conséquent, la qualité de service de la présentation en xhtml+smil est réduite par rapport à la version initiale.

Pour conclure, ce type de conversion conserve la présentation initiale dans quasiment la totalité de ses aspects. Seule la qualité de service est dégradée et la flexibilité sur la durée est supprimée. Ces deux fonctionnalités n'ont aucun impact critique sur la présentation finale du scénario multimédia. En effet, la solution de remplacement est un cas particulier des solutions de formatage engendrées par ces deux fonctionnalités.

5.2.3. De Madeus 2.0 vers le profil SMIL 2.0

La deuxième conversion s'effectue à partir de document Madeus 2.0 vers des documents SMIL 2.0. Cette conversion est relativement simple puisque ces deux langages sont relativement proches. Néanmoins, la difficulté est la conversion des opérateurs spatiaux, des relations et de la flexibilité. Le formatage de ces éléments est une opération complexe qui ne peut pas être effectuée par un langage de transformation. C'est pourquoi la conversion de Madeus vers SMIL 2.0 nécessite que le document initial soit d'abord formaté. À partir du résultat du formatage il est alors facile de transformer le document (cf. figure 45).

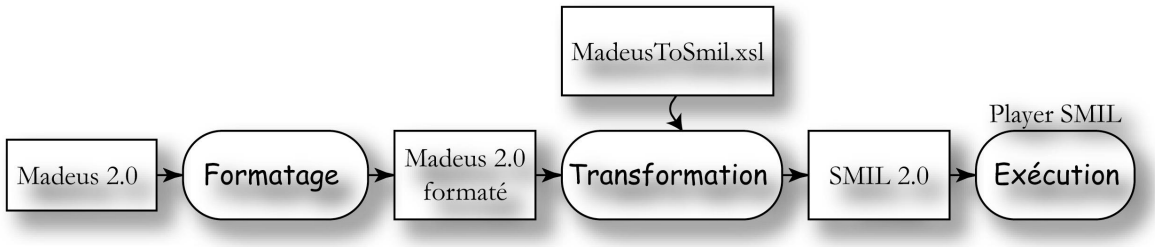


Figure 45. Étapes de conversion d'un document Madeus 2.0 vers un document SMIL 2.0.

Cette transformation montre les limites du processus de transformation quand il s'agit d'interpréter la sémantique d'une présentation, par exemple pour calculer les positions absolues des régions pouvant être reliées entre elles. Il est néanmoins possible d'invoquer une fonction externe au langage XSLT mais au prix d'une complexité et d'un surcoût de conception qui peut devenir ingérable par un auteur.

5.2.4. De Madeus 2.0 vers le profil SMIL Basic

Le profil SMIL basic définit un ensemble de modules de base pour spécifier une présentation multimédia adaptée à des terminaux caractérisés par de faibles ressources. Ce profil est extensible, c'est-à-dire que le nombre de modules employé peut varier : d'autres modules que ceux de base peuvent être ajoutés.

Par rapport à Madeus et au profil SMIL 2.0, SMIL Basic comporte que très peu de fonctionnalité. Certaines de ces fonctionnalités peuvent être obtenues en partant d'un document formaté. Cependant, cela ne suffit pas car un module comme celui de l'animation ne peut être représentée en SMIL Basic. Pour certains scénarios multimédia, une solution est

de filtrer les animations sans entraîner de conséquence sur « l'utilisabilité » de ce scénario. Cette solution ne peut malheureusement pas se généraliser. Par exemple, en supposant qu'un scénario multimédia consiste à montrer la direction à prendre par un véhicule à travers une animation, il n'existe aucune autre solution pour transformer ce document de telle façon qu'il soit exploitable. En effet, il est nécessaire dans ce cas d'avoir recours à des explications textuelles très difficiles à induire uniquement à partir de la spécification sur l'animation. Cela montre qu'il n'est pas toujours possible de présenter des documents Madeus 2.0 et SMIL 2.0 sur des PDA pour lesquels le profil SMIL Basic est en particulier destiné. Par conséquent, pour générer une présentation SMIL Basic, il est nécessaire que le document source ne contienne pas dès le départ ce genre de fonctionnalité. Ce constat conduit à la définition de règles de conception de documents adaptables.

5.2.5. Bilan

Les trois conversions ont été implémentées en utilisant le langage XSLT (cf. [Villard 01a] pour la première et [Villard 01b] pour la dernière) dans les limites citées ci-dessus. Chaque transformation consiste en une seule feuille de transformation. La taille de chacune de ces transformations représente respectivement 600 lignes, 250 lignes et 200 lignes.

La seconde expérimentation montre les limites du processus de transformation pour interpréter la sémantique d'une présentation. Cette limite a un impact sur les possibilités d'adaptation avec le système actuel. Il n'est en effet pas possible d'exprimer des besoins d'adaptation qui dépendent à la fois du domaine et de la présentation. Par exemple, il n'est pas possible d'optimiser l'espace occupé par les items de liste d'un transparent en respectant la contrainte d'avoir des caractères de taille minimum et de présenter les items sur une autre page si la taille du transparent est dépassée, ces deux opérations étant effectuées par deux modules séparés le découpage ne peut en effet se faire que par le processus de transformation, tandis que l'optimisation de l'espace ne peut s'effectuer que par le processus de formatage.

Dans la dernière expérimentation, nous avons montré les limites d'adaptation d'un document Madeus 2.0. Une des causes est le manque d'expressivité de ce modèle pour capturer les intentions de l'auteur. Sans une modélisation formelle de ses intentions, il est fort probable qu'une solution performante pour adapter un document aux multiples contextes utilisateur soit hors de portée. Dans le cas de l'exemple d'utilisation des animations pour indiquer le chemin à suivre, le système adaptable doit être en mesure d'interpréter au moins ce qu'est un chemin et une direction. C'est dans l'ambition de définir ce type de modèles que s'inscrit les travaux de modélisation des tâches utilisateur dans le système Aurora (cf. chapitre 3 section 4.2.2). En suivant cette logique, les documents vont devenir de plus en plus abstraits. L'usage d'un processus de transformation est alors indispensable pour compenser l'important fossé créé par cette abstraction pour produire les représentations finales.

6. Conclusion

Dans ce chapitre, nous avons défini l'architecture générale d'un système adaptable qui supporte les classes de documents multimédias et qui répond au besoin d'adapter de tels documents aux contraintes des terminaux. Nous avons identifié les langages impliqués dans ce système ainsi que les trois processus qui permettent de produire une présentation adaptée : la transformation, le formatage et l'exécution. Lors des expérimentations, nous avons montré la pertinence de ce système pour répondre aux objectifs que nous nous sommes fixés. Nous

avons aussi montré les limites d'une telle approche. Enfin, au cours de ce chapitre, nous avons mis en évidence à travers plusieurs exemples le fait que chacun des composants du système adaptable avait un rôle bien déterminé et complémentaire.

Ce système remet en cause la méthodologie actuelle de conception de documents multimédias. Afin de rendre le contenu accessible à différents contextes utilisateur, il nous paraît indispensable de le séparer de ses multiples représentations, et d'utiliser un processus de transformation pour réunir ces informations. La surcharge de travail introduite par la conception des feuilles de transformation doit être accompagnée d'une aide au moyen d'outils d'édition que nous proposons dans le chapitre suivant.

Chapitre 5

Outil d'aide pour l'édition de documents adaptables

Dans le chapitre précédent, nous avons décrit un système de production de présentations adaptées au contexte de l'utilisateur. L'objectif de ce chapitre est de proposer un outil qui permet d'assister l'auteur pour éditer deux type d'informations essentielles qui interviennent dans ce système : le document source et les feuilles de transformation. Pour cela nous commençons par définir les besoins pour concevoir cet outil d'édition. Puis nous décrivons comment est réalisée l'édition dans ce système du document source puis des feuilles de transformation.

1. Introduction

1.1. Motivation

L'édition de documents adaptables et de documents respectant un modèle comprend l'ensemble des informations constituant le document source et la spécification de la ou les transformations (cf. figure 46). Dans la suite de ce chapitre, nous désignons par le terme document deux catégories d'informations : les informations pérennes (document source et feuilles de transformation) et les informations temporaires (document de présentation et document formaté). Pour l'auteur, ce sont les informations pérennes qui le concernent.

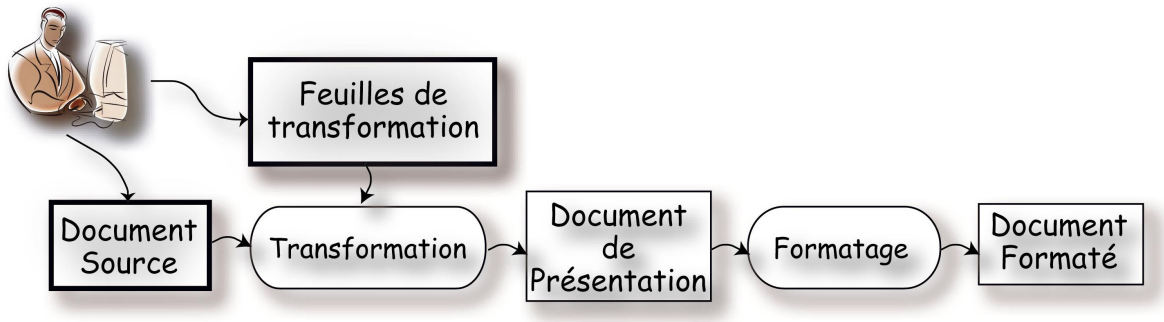


Figure 46. Implication de l'auteur dans le processus d'édition.

Les deux ensembles d'informations pérennes peuvent être écrits en utilisant les outils d'édition de documents XML présentés dans le chapitre 2. Pour être plus précis, pour éditer le document source, seuls les outils génériques peuvent être utilisés (XMLSpy, XMLPro, XMetal, etc.). Cependant, sans éditeur spécialisé, la tâche d'édition d'un document est fastidieuse, relativement difficile d'accès et surtout inefficace. Seules des fonctions d'édition de bas niveau peuvent être offertes, c'est-à-dire celles qui manipulent de façon élémentaire le contenu et la structure du document. Par exemple, pour insérer une table dans un document de type Docbook en utilisant un outil générique, cinq opérations d'insertion d'éléments sont nécessaires. Au mieux, l'outil d'édition peut utiliser la définition formelle du modèle pour insérer les éléments obligatoires, ce qui réduit sensiblement le nombre d'opérations d'édition.

Dans un système d'édition spécialisé, cette insertion s'effectue en une seule opération : « insérer table ». Ce type d'opération est, entre autres, offert par le module d'édition de tables HTML inclus dans Emacs.

De façon générale, il est peu probable de trouver un outil d'édition directement adapté à un modèle de document source donné. En effet, des modèles sont souvent définis par des organismes ou des personnes tierces qui ne sont pas impliqués dans le développement d'outils d'édition. Même pour des modèles de document largement répandus, comme Docbook, il n'existe pas d'outil d'édition dédié (KOffice [KOffice] ne propose qu'une fonction d'exportation).

En ce qui concerne l'édition de feuilles de transformation XSLT, il existe des outils d'édition spécialisés (VisualXSLT, XSLDebug, Stylus Studio, etc.). Cependant, comme nous l'avons montré dans le chapitre 2, du point de vue du génie documentaire, ces outils souffrent de plusieurs lacunes. En effet, la majorité de ces outils s'apparente à des outils de programmation classique : l'auteur écrit les feuilles XSLT sous forme textuelle et peut tester son code en utilisant un débogueur. Ce type d'outil est adapté, car le seul possible, pour écrire des transformations complexes dont le résultat n'est pas un document de présentation. Par contre, lorsqu'il s'agit de produire des présentations comme SVG ou SMIL, l'utilisation de ce type d'outils rend la tâche de conception plutôt fastidieuse. Un seul éditeur, à notre connaissance, permet de créer des transformations de façon conviviale et interactive (<xsl>Composer, cf. chapitre 2 section 4.2.2). Mais à ce jour, cet outil est loin de prendre en compte tous les traits du langage XSLT. De plus, il est conçu pour éditer des transformations qui produisent uniquement des documents HTML (cf. chapitre 2).

Dans ce chapitre, nous proposons un outil auteur qui couvre une partie des besoins cités ci-dessus. Le premier objectif est d'aller plus loin que les outils d'édition de document XML existants. Nous verrons que l'utilisation d'un processus de transformation est le composant clé pour répondre à cet objectif. Le second défi est de pouvoir éditer des feuilles de transformation de façon conviviale et interactive. La simplicité et la convivialité de ces deux fonctions d'édition sont obtenues à partir d'une édition sur le document formaté (cf. figure 46). Donc notre proposition d'édition consiste à regrouper ces deux aspects dans un outil commun.

1.2. Identification des besoins

La plupart des systèmes d'édition sont composés des fonctionnalités suivantes :

- un ensemble de fonctions de présentation pour visualiser le document. Dans un contexte d'édition, à cause de la complexité des documents multimédias, il est nécessaire d'offrir à l'auteur des *vues* partielles du contenu de ces documents [Tardif 00]. Ces vues partielles montrent un aspect particulier du document, comme le placement temporel des médias et les attributs d'un média. Le rôle des fonctions de présentation est de montrer le document selon ces différents aspects ;

- un ensemble de fonctions d'édition. Plusieurs niveaux peuvent être distingués. Dans le cas d'édition de documents XML, trois niveaux de fonction d'édition sont identifiés :

  - les fonctions d'édition de base sont celles définies par le standard DOM. Ces fonctions sont en particulier l'ajout et la suppression d'un élément, la modification de la valeur d'un attribut, ou la modification d'un contenu ;

  - les fonctions d'édition intermédiaires sont une composition des fonctions de base, comme le déplacement de nœuds, la copie et le collage de fragment de document, etc. Ces fonctions sont définies indépendamment du modèle de document sous-jacent ;

  - les fonctions d'édition de haut niveau sont aussi une composition des fonctions de base. Cependant ces fonctions font intervenir la sémantique du document édité. Par exemple, pour un document de type SMIL, une fonction de haut niveau est, par exemple, de glisser le nom d'un fichier contenant les données d'une image dans la présentation. Le résultat en terme d'opération de base est l'ajout d'un élément de type image, d'un attribut référençant le fichier glissé et d'une région dont le bord haut et gauche correspond à la position du lâché de fichier. Pour la même opération d'édition, le résultat est différent lorsque le document est de type Docbook : les éléments de type `mediaobjet`, `imageobject` et `imagedata`, ainsi que l'attribut `fileref` sont générés ;

- de services de vérification. Ces services vérifient que le document édité est conforme à un modèle. Le cas idéal est que l'ensemble des contraintes d'un modèle puisse être exprimé formellement. Or, pour le moment, seules des contraintes de nature logique peuvent être spécifiées par les méta-modèles existants. En effet, ils ne prennent pas en compte des contraintes d'ordre linguistique, telles que les contraintes sur l'orthographe, sur la grammaire, ou sur la cohérence d'une phrase. Ils ne prennent pas en compte non plus des contraintes de sémantique de présentation. Il est donc nécessaire d'étendre les éditeurs par des outils de vérification spécifiques au modèle de document ;

- d'un ensemble de fonctions de gestion documentaire. Ces fonctions portent sur le document dans sa globalité ; elles n'agissent pas sur le contenu du document. Par exemples, ces fonctions sont l'ouverture, la fermeture, la comparaison, ou l'archivage de documents ;

- d'une interface graphique. En particulier, c'est à partir de l'interface graphique que sont accomplies les fonctions d'édition (cf. figure 47). De façon plus précise, l'interface graphique offre plusieurs *actions* accessibles par l'auteur. Ces actions se traduisent par une ou plusieurs *opérations d'édition* sur un document. La conception d'un système auteur nécessite de définir à la fois les actions possibles et leur traduction en opérations d'édition. Certaines actions se traduisent directement en une seule opération d'édition, comme l'activation de la commande « ajout d'un élément » se traduit directement par l'opération d'édition « ajout d'un élément » dans le document. Cependant, ce n'est pas toujours le cas lorsque que des actions plus complexes sont considérées, et en particulier lorsqu'il s'agit de mettre à jour un programme XSLT.

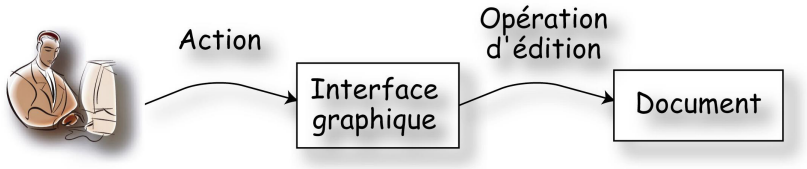


Figure 47. La fonction d'édition se décompose en une action auteur se traduisant en une ou plusieurs opérations d'édition sur le document.

En plus des besoins cités préalablement, nous en avons identifiés plusieurs relatifs à l'édition de documents adaptables. Ces besoins sont les suivants :

- le besoin d'associer les paramètres d'adaptation à un résultat sur la présentation. Bien que certains paramètres soient pris en compte de façon automatique, ceux liés au domaine (cf. section 2.3 chapitre 3) nécessitent l'intervention de l'auteur ;
- le besoin de pré-visualiser et de naviguer à travers les résultats de l'adaptation. Le nombre potentiellement important de solutions d'adaptation ne permet pas à l'auteur de les vérifier toutes, néanmoins il peut vérifier le résultat sur certains paramètres d'adaptation qu'il juge critique ;
- le besoin d'éditer conjointement à la fois le document source et la spécification de la transformation. Dans le système adaptable tel que nous l'avons défini, ces deux spécifications sont indissociables.

En analysant l'ensemble des fonctionnalités décrites dans cette section, deux catégories peuvent être définies : les fonctionnalités indépendantes d'un modèle de document et celles qui, au contraire en sont dépendantes. Celles qui sont indépendantes peuvent être intégrées directement dans notre système d'édition. Par contre, les autres fonctionnalités ne peuvent être offertes puisqu'elles ne peuvent pas être connues *a priori*.

1.3. Architecture générale de l'outil d'édition

Les besoins identifiés dans la section précédente nous a conduit à définir l'architecture et le processus d'une session d'édition illustrés figure 48. L'utilisateur modifie le document source et les feuilles de transformation à travers un ensemble de vues. Chaque vue montre le document associé sous une forme particulière. De plus, certaines vues permettent un certain nombre d'actions destinées à l'utilisateur qui se traduisent en opérations d'édition sur le document. Pour simplifier, nous distinguons deux catégories de vues : les *vues source* et les *vues cible*. Les vues source visualisent le document source n'ayant subi aucune transformation, ainsi que les feuilles de transformation. Les vues cible, au contraire, montrent le document source après la phase de transformation, c'est-à-dire qu'elles montrent le document de présentation formaté (ou non).

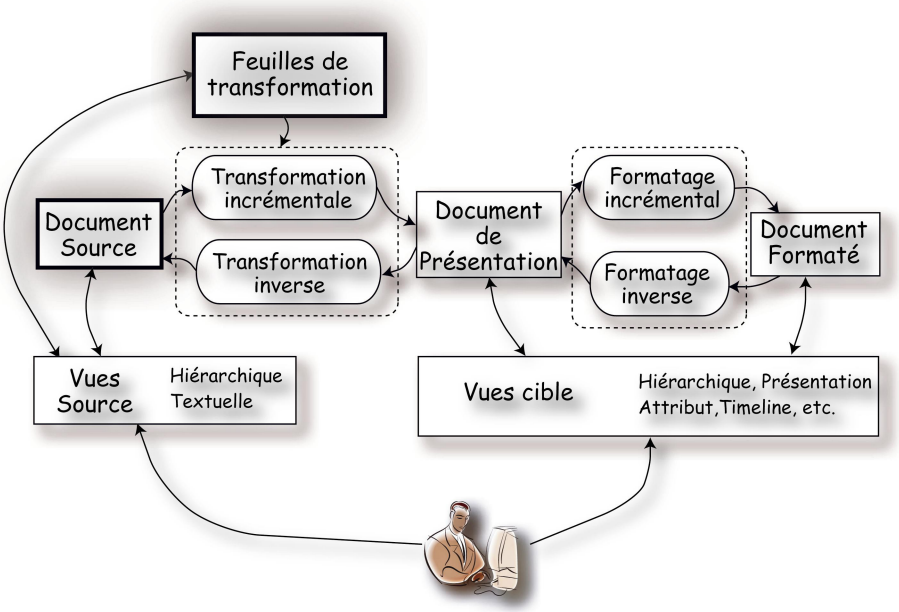


Figure 48. Architecture générale du système d'édition.

Les principales vues de notre système d'édition sont les suivantes :

- la vue textuelle qui affiche le document sous la forme XML. Cette vue est utilisée pour montrer le document source et le document de présentation. Elle permet aussi d'éditer le document de façon textuelle ;
- la vue hiérarchique visualise le document source et de présentation en montrant ses relations hiérarchiques. Elle offre les deux premières catégories de fonctions d'édition identifiées dans la section précédente. Nous verrons plus en détail cette vue dans les sections 2.1 et 4.4.1 ;
- la vue attribut visualise les attributs des éléments du document. Cette vue complète la vue hiérarchique en offrant une interface utilisateur adaptée à chaque type d'attribut ;
- la vue de présentation montre l'exécution du document de présentation. Cette vue s'applique uniquement au document de présentation formaté. Au niveau de l'édition, elle permet de modifier l'organisation spatiale des objets du document, comme les régions pour les documents Madeus et SMIL. Elle permet aussi d'éditer le contenu de certains médias, en particulier le média texte ;
- la vue timeline permet d'afficher et de manipuler des informations de nature temporelle.

Quelle que soit la vue à partir de laquelle l'auteur interagit, le résultat de l'action effectuée par l'auteur est la modification du document source et/ou d'une feuille de transformation. Cependant, selon le document modifié, le processus sous-jacent pour répercuter les modifications sur le document source et les feuilles de transformation varie. L'auteur peut en effet agir sur trois documents :

- le document source : dans ce cas la modification est directement répercutée sur le document source ;

- le document de présentation : comme le document de présentation est généré à partir du document source, il est nécessaire de modifier les données de génération : soit le document source, soit les feuilles de transformation. La détermination de la modification à effectuer est calculée par un processus appelé *transformation inverse* ;

- le document formaté : lorsque l'auteur agit sur le document formaté, pour modifier le document source ou les feuilles de transformation, il est nécessaire de parcourir toute la chaîne de présentation dans le sens inverse. À partir du document formaté sont retrouvées la ou les modifications à effectuer sur le document de présentation grâce au processus de *formatage inverse*. Ensuite le processus est identique à celui décrit ci-dessus.

Après une opération d'édition, ce processus inverse ne suffit pas. En effet, la modification sur le document source ou la feuille de transformation peut entraîner plusieurs modifications du document formaté, autres que celles effectuées par l'auteur. Par exemple, dans la présentation de transparents du chapitre précédent, le fait de modifier un titre dans la table des matières entraîne aussi la modification du titre présenté dans le transparent lui-même. De plus, l'auteur peut modifier l'entité titre parmi ses multiples représentations, en particulier la vue source hiérarchique. Dans un contexte d'édition interactive par manipulation directe, la mise à jour des représentations de « l'objet d'intérêt » (celui en cours d'édition) doit se propager le plus rapidement possible [Shneiderman 97]. Pour cela, nous nous sommes fixé comme but de ne calculer que les informations qui sont incohérentes en utilisant des algorithmes incrémentaux (cf. chapitre 2). Au niveau de la transformation, la mise à jour est effectuée par le processus de *transformation incrémentale*. Au niveau du formatage, la mise à jour est réalisée par le processus de *formatage incrémental*. Alors que les processus de formatage incrémental et inverse sont bien connus (cf. chapitre 2), la réalisation des processus de transformation incrémentale et de transformation inverse n'a pas encore, à notre connaissance, de solutions pour un langage de transformation complexe comme XSLT. C'est pourquoi, dans ce mémoire, nous nous focalisons sur ces deux processus, le premier étant étudié dans la section 2.2.2 de ce chapitre, tandis que le second est étudié dans le chapitre 6.

1.4. Plan de la suite

Dans de ce chapitre, nous détaillons les actions d'édition possibles à travers les vues de notre système d'édition et quelles sont les opérations d'édition à effectuer. Nous déterminons d'abord les actions et les opérations d'édition concernant la modification du document source à partir d'une vue source et d'une vue cible. Nous verrons que les actions offertes par les vues cibles ne sont pas suffisantes. C'est pourquoi nous proposons et décrivons ensuite un mécanisme de génération d'éditeur spécialisé. Dans la section 4, l'édition de la transformation est détaillée. Nous terminons par un bilan sur l'édition à travers le système d'édition décrit dans ce chapitre.

2. Édition du document source

Nous avons vu que l'édition de document XML s'effectue à travers une visualisation textuelle et hiérarchique. Ce premier mode d'édition est très utile car ces vues permettent d'avoir une représentation exacte et directe du source XML. Le second mode d'édition est d'éditer le document source à partir d'une vue du document de présentation formaté ou non. Ces deux modes d'édition sont étudiés successivement dans la suite de cette section.

2.1. Vue source

Notre outil fournit deux vues pour éditer directement le document source : la vue textuelle et la vue hiérarchique. Ces vues sont classiques et sont disponibles dans la majorité des outils d'édition génériques de document XML. Cependant, rares sont les outils qui proposent une édition directe dans la vue hiérarchique. En effet, dans la plupart des outils existants, la vue hiérarchique ne sert que de support pour naviguer dans le document [Morphon], ou ne montre que les éléments sans leurs attributs qui sont visualisés dans une vue à part [XMLPro V2, EditMLPro, XMetal2.0]. Le problème rencontré dans ces outils est que les attributs sont présentés en dehors de leur contexte de définition. Par conséquent, ils ne permettent pas d'avoir une vision globale du document puisque seul un fragment est présenté à la fois, ce qui a notamment pour conséquence d'augmenter la charge de travail de l'auteur (cf. chapitre 2 section 4.1). Seuls quelques outils comme XMLSpy [XMLSpy] et Amaya [Amaya] offrent une édition directe dans la vue hiérarchique. Toutefois, la vue hiérarchique de XMLSpy ne répond pas au critère d'homogénéité d'une interface utilisateur (cf. chapitre 2 section 4.1). Par contre, dans Amaya la vue hiérarchique correspond mieux à nos attentes et notre vue s'inspire donc fortement de celle d'Amaya.

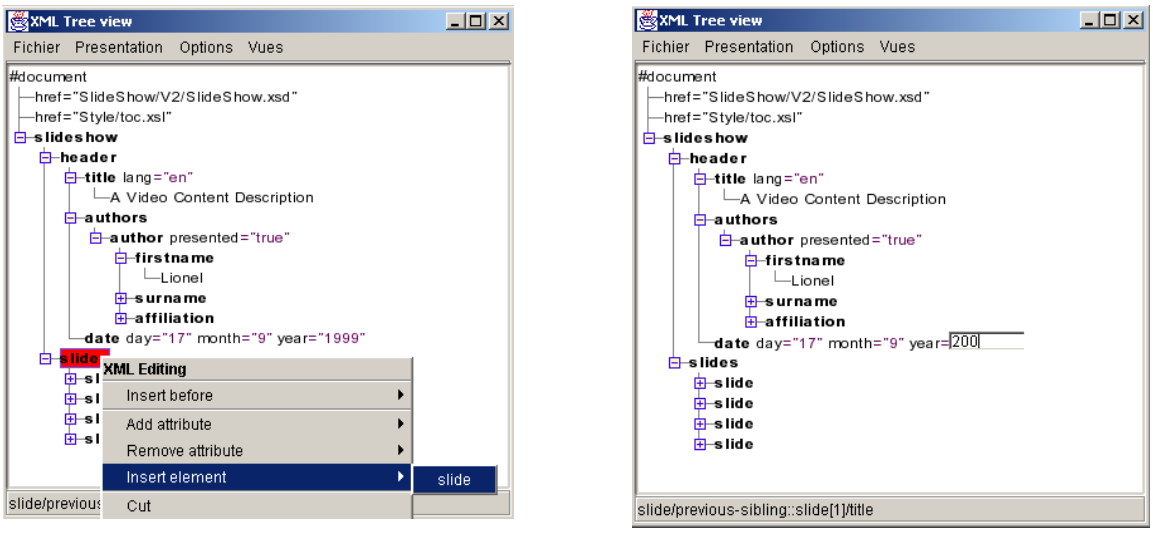


Figure 49. Exemple d'édition dans la vue hiérarchique. À gauche est ajouté un élément de type slide. À droite la valeur de l'attribut year est modifiée.

La vue d'édition hiérarchique que nous proposons est illustrée figure 49. Les éléments et les textes sont présentés sous forme hiérarchique. Les attributs des éléments sont présentés à droite de l'élément sur une ou plusieurs lignes. L'édition de la valeur des attributs et des textes se fait directement dans cette vue. L'ajout et la suppression de nœuds (attribut, texte, élément, commentaire et instruction de programme) se font à travers un menu déroulant. D'autres fonctions de plus haut niveau sont offertes. On peut citer en particulier le déplacement de nœuds, les actions de copier et de coller, ou encore le changement de nom d'un élément. Chaque action effectuée dans cette vue se traduit directement en opération d'édition équivalente sur le document source.

Au delà de l'édition d'une instance XML, nous verrons plus loin l'importance de disposer dans une même vue à la fois les éléments, les textes et les attributs pour l'édition d'expressions XPath nécessaires pour créer des feuilles de transformation (cf. section 4.4.1).

Les vues source permettent d'éditer simplement un document XML en couvrant les deux premiers niveaux de fonctions d'édition identifiés dans la section 1.2. Cependant ce type d'édition est souvent frustrante car les répercutions sur la ou les présentations ne sont pas immédiatement perceptibles. Le premier pas pour résoudre ce problème est d'offrir des aperçus du résultat. Après chaque modification du document source, le résultat de la transformation est mis à jour de façon incrémentale : l'auteur peut ainsi voir promptement les conséquences de ses modifications. Cependant, dans le cas où le résultat ne lui convient pas, il doit identifier les éléments du document source fautifs et les corriger, ce qui peut être une tâche ardue lorsque le document de présentation est complètement différent du document source, ce qui est souvent le cas pour des présentations multimédias.

À notre avis, l'approche la plus efficace consiste à mettre à jour directement le résultat sur la présentation. C'est pourquoi nous proposons une édition directe à partir des vues cible que nous détaillons dans la section suivante.

2.2. Vue cible

2.2.1. Principe

Lorsque l'auteur modifie le document source à partir d'une vue cible, les actions qu'il effectue nécessitent deux étapes de traduction (cf. figure 50). La première étape est la traduction de l'action vers une ou plusieurs opérations d'édition. Ensuite ces opérations d'édition sont traduites en une ou plusieurs modifications sur le document source grâce au processus de transformation inverse.

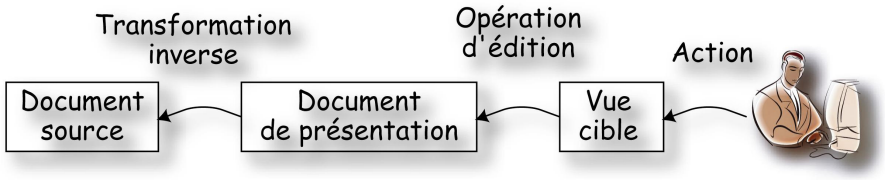


Figure 50. Processus d'édition du document source à travers une vue cible.

L'expressivité en terme de modifications possibles sur le document source est conditionnée principalement par la fonction de transformation inverse. C'est pourquoi nous commençons pas détailler ce processus dans la section suivante. Nous verrons ensuite les fonctions d'édition offertes par la vue de présentation pour modifier le document source.

2.2.2. Transformation inverse

Le problème de la transformation inverse est de retrouver les nœuds du document source qui affecte ou qui sont à l'origine d'un nœud cible. Cette recherche s'effectue en deux étapes (cf. figure 51). La première étape consiste à retrouver le nœud (ou instruction) de la spécification XSLT qui a généré le nœud cible. Cette instruction est obligatoirement une instruction de production, comme les instructions `value-of`, `element` et `attribute`. La seconde étape consiste

à retrouver les nœuds source à partir de cette instruction. Ce processus de recherche dépend de la catégorie du nœud source.

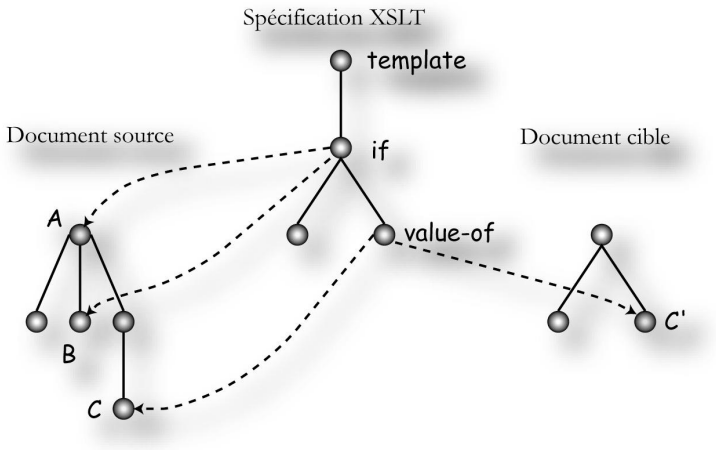


Figure 51. Catégories des nœuds source. Les noeuds A et B sont des nœuds indirects. Le nœud C est un nœud direct.

Sur la figure 51 sont représentés deux catégories de nœuds source selon leur utilisation dans la feuille de transformation : les nœuds directs et les nœud indirects. Les nœuds source sont dits *directs* lorsqu'ils sont liés à un nœud cible par l'intermédiaire d'une seule instruction XSLT. Dans le cas contraire, les nœuds sont dits *indirects*. Un exemple de nœud direct est le nœud C qui est lié au nœud C' par l'instruction value-of. Le processus de recherche est alors très simple puisqu'il suffit de parcourir les liens entre l'instruction de production et les nœuds source sélectionnés par l'expression associée à l'instruction. En ce qui concerne les nœuds indirects, ils sont retrouvés en parcourant les nœuds ascendants de l'instruction et en traversant leurs liens vers le document source. Dans l'exemple, le nœud C' est lié indirectement avec les nœuds A et B utilisés lors de l'évaluation de l'instruction conditionnelle if. Pour un même nœud cible des nœuds sources directs et indirects peuvent être associés.

Le processus de transformation inverse nécessite de conserver les liens entre le document source et la feuille de transformation, ainsi que les liens entre la feuille de transformation et le document cible. Il faut noter que lors de l'exécution de la feuille de transformation XSLT, une même instruction peut être exécutée plusieurs fois. Ces exécutions diffèrent selon le nœud source, appelé *nœud contextuel*, qui a servi lors de l'instanciation de la règle dans laquelle l'instruction est déclarée. Par conséquent, chaque lien dépend du nœud contextuel. De façon formelle, un lien d'une instruction XSLT vers le document source est un couple (nœudSourceContextuel, {nœudSource}). Un lien d'un élément cible vers le document XSLT est un couple (nœudSourceContextuel, noeudCible).

Dans notre système d'édition, la réalisation de la transformation inverse repose sur l'arbre d'exécution construit par le processeur de transformation incrémental (cf. chapitre 6). Cet arbre représente le flot d'exécution d'une spécification XSLT. Chaque nœud de cet arbre correspond à l'exécution d'une instruction pour un nœud contextuel donné. Les liens sont stockés sur chacun de ces nœuds d'exécution et non pas sur le document cible et/ou le document source. Ce choix d'implantation est motivé essentiellement par le fait que les

transformations peuvent être chaînées (cf. chapitre 4 section 5) et que les documents cible intermédiaires ne sont pas toujours stockés en mémoire.

Techniquement, la transformation inverse est utilisée pour identifier les nœuds du document source à partir d'un nœud du document cible. Au niveau applicatif, la transformation inverse sert lors de la mise à jour et de la sélection de nœuds du document source lorsqu'une action est accomplie dans une vue cible. Nous étudions la première application citée dans la section suivante. Nous verrons une autre application de la première étape de la transformation inverse lors de la description de l'édition de la feuille de transformation (cf. section 4).

2.2.3. Application de la transformation inverse à l'édition du document source

Nous avons vu que chaque action effectuée dans une vue cible se traduit en une ou plusieurs opérations d'édition. Les actions de chacune des vues cible sont dédiées à la modification du document de présentation, comme le déplacement d'une région ou l'ajout et la suppression d'un média dans la vue de présentation. Dans certains cas, ces actions se traduisent par la modification du document source. Nous analysons ces deux types d'action afin d'observer les conséquences sur le système d'édition.

Considérons la spécification suivante :

```
1.  <xsl:template match="section">
2.      <madeus:region left="{@left + 10}" top="{@top + 20}"/>
3.  </xsl:template>
```

Dans cet exemple, la position de la région dépend de deux attributs source left et top. L'action de déplacement d'une région dans la vue d'exécution peut se traduire par la modification de ces deux attributs source. Une autre possibilité est de modifier directement la feuille de transformation en remplaçant les deux expressions {@left + 10} et {@top + 20} par la position effective de la région. Cette ambiguïté nous conduit à définir deux modes d'édition : l'édition du document source et l'édition de la spécification de la transformation. C'est à l'auteur de choisir quelle entité il veut modifier. Dans cette section nous considérons uniquement le premier mode, le second mode est étudié dans la section 4.

L'exemple ci-dessus montre un cas simple de traduction. Un exemple plus compliqué est la suppression d'un média produit de la façon suivante :

```
1.  <xsl:if test="@showTitle='true' and title">
2.      <madeus:text><xsl:value-of select="title"/></madeus:text>
3.  </xsl:if>
```

Dans cet exemple, le média texte est généré lorsque l'attribut showTitle contient la valeur vraie et lorsque le nœud contextuel (cf. section précédente) contient un fils de type title. Lorsque l'élément est supprimé depuis la vue d'exécution, deux cas sont alors possibles pour éviter que le média texte soit généré : soit changer la valeur de l'attribut showTitle ou soit supprimer l'élément title. Même à travers ce petit exemple, la traduction est ambiguë. Cette ambiguïté est croissante lorsque les spécifications sont plus complexes. Dans ce contexte, il est difficile de lever l'ambiguïté. En plus de ce problème d'ambiguïté, l'ajout ou la suppression d'un média pour modifier un concept relatif au document source pose le problème de

l'intention de l'auteur et de l'interprétation faire par le système : le critère de compatibilité de Scapin et Bastien n'est en effet pas respecté (cf. chapitre 2 section 4.1).

Ainsi, pour concevoir un système d'édition cohérent et compatible, nous ne permettons que les actions de modification du contenu d'un objet textuel lorsque cela correspond à la modification du contenu d'un élément ou la valeur d'un attribut. Cette traduction préserve l'image que l'auteur se fait d'une telle action sur le document. Elle suppose que le système est capable de calculer l'expression inverse. Avec les contraintes imposées sur les actions, ce calcul peut facilement être réalisé, puisqu'à chaque constructeur d'une expression correspond un constructeur inverse.

2.3. Expérimentation : l'éditeur SlideShow

Les différents concepts présentés dans cette section ont été expérimentés pour des documents SlideShow. Cet éditeur repose sur la présentation générée par les feuilles de transformation décrites dans le chapitre précédent (cf. figure 33). Sans aucune modification de la spécification, l'auteur peut modifier le titre des transparents (cf. figure 52), ainsi que le contenu des listes d'item. Ces actions sont celles décrites dans la section 2.2.3 de ce chapitre.

Figure 52. Exemple d'édition du titre du second transparent. On peut noter la mise à jour du titre à droite.

L'éditeur ainsi obtenu est relativement limité. En particulier, il n'est pas possible d'ajouter ou de supprimer des transparents, d'ajouter un item de liste lorsque l'auteur appuie sur la touche entrée dans un item de liste, etc.

2.4. Limites

Nous avons vu que la modification du document source à partir d'une vue cible ne permet pas d'effectuer des modifications importantes. L'ajout et la suppression de nœuds ne sont pas possibles à cause notamment de l'ambiguïté liée à ces actions. Pour permettre une édition plus complète dans ces vues, il est nécessaire d'étendre les actions existantes par des actions spécifiques au modèle du document source. Puisque ce modèle n'est pas connu *a priori*, ces actions sont générées à partir du modèle de document. De plus, comme chaque action est activée à travers une interface graphique, il est nécessaire de générer aussi l'interface utilisateur. Nous étudions ces aspects dans la section suivante.

3. Génération d'un éditeur dédié

Un moyen d'augmenter à la fois l'efficacité et la convivialité de l'édition est d'utiliser le processus de transformation pour générer un éditeur dédié à un modèle de document. Pour cela, il est nécessaire que le langage résultat de la transformation puisse représenter les différents besoins énoncés dans la section 1.2. Le modèle de document multimédia Madeus 2.0, utilisé dans notre système d'édition ne permet ni l'ajout d'action utilisateur autre que la navigation, ni la personnalisation de l'interface graphique. Nous l'avons donc étendu par un langage de description d'interface graphique. Ce langage doit pouvoir représenter des interfaces graphiques personnalisées à chaque modèle de document. Il doit aussi permettre de spécifier des actions d'édition spécifiques au modèle de document.

Dans la section suivante, nous analysons les langages de description d'interface graphique existants afin d'en choisir un qui répond aux besoins énoncés ci-dessus. Nous décrivons ensuite les expériences que nous avons menées sur le modèle de document Docbook.

3.1. Choix d'un langage

En étudiant les langages de description d'interface graphique, nous en avons recensé deux : le langage UIML [UIML 00] et le langage XUL [XUL 01]. Nous donnons les principales caractéristiques de ces langages ci-dessous.

3.1.1. Le langage UIML

Ce modèle de description d'interface graphique a été défini par la société Harmonia. L'objectif de ce langage est de pouvoir décrire une interface graphique indépendante de la plate-forme hôte. Pour cela, l'interface graphique est décrire en termes génériques et sa correspondance avec le langage de programmation et le système sous-jacent est réalisée par l'intermédiaire d'une description de styles.

Les principaux éléments d'une description UIML sont les suivants :

- l'élément `structure` : cette partie définit la structure des composants graphiques, appelés `part`, de l'interface. Chaque élément `part` est caractérisé par un nom d'instance et un nom de classe. Le nom d'instance identifie de façon unique le composant graphique et le nom de classe identifie le composant comme appartenant à une classe ;
- l'élément `style` : cette partie décrit le style (position, alignement, polices de caractères, couleur de fond...) pour chaque classe de composants graphiques ou pour des instances individuelles. Le vocabulaire de style est spécifique au terminal cible de l'interface ;
- l'élément `content` : le contenu des composants graphiques est spécifié de façon séparée. Cet élément facilite l'internationalisation de l'interface utilisateur ;
- l'élément `behavior` : cet élément permet de décrire les évènements de l'interface. Un événement est composé d'une ou plusieurs conditions et d'une ou plusieurs actions :
    - l'élément `condition` : une condition est vérifiée soit lorsqu'un événement particulier est arrivé, comme l'activation du bouton de la souris, ou soit lorsqu'une propriété de style évolue vers une valeur particulière ;

  - l'élément `action` : trois types d'actions sont possibles : changement de propriétés de style du composant, appel à une méthode externe, et déclenchement d'événements ;

- l'élément `peers` : la description des différentes implémentations des composants graphiques selon la plate-forme cible est contenu dans cet élément. La même interface utilisateur peut ainsi être adaptée sur différentes plates-formes (station de travail, PDA, téléphone cellulaire, etc.).

En plus de ces éléments, UIML propose plusieurs vocabulaires, chacun étant spécifique à une plate-forme donnée. Dans ce vocabulaire sont inclus les définitions de propriété de style, d'événements et de noms de composants graphiques. Actuellement, les vocabulaires définis sont dédiés aux langages HTML 3.2, WML, Java AWT 1.3 et Java Swing 1.3.

3.1.2. Le langage XUL

XUL a été créé pour faciliter et accélérer le développement du navigateur Mozilla [Mozilla]. L'objectif de XUL est similaire à UIML, c'est-à-dire que la spécification de l'interface graphique doit être multi plate-forme. L'approche est cependant différente, puisque XUL définit un seul vocabulaire spécifique pour spécifier les composants graphiques. La correspondance avec la plate-forme est réalisée implicitement par l'implantation de XUL.

Dans XUL, une interface graphique est décrite en trois parties :

- la partie `content` : dans cette partie sont décrits tous les constituants d'une interface graphique, c'est-à-dire les composants graphiques et les événements. Les composants graphiques sont représentés par des noms spécifiques, comme `button`, `scrollbar`, `menu`, `menuitem`, etc. Le modèle événementiel est similaire au modèle HTML : la condition est l'activation d'un événement (`onClick`, `onFocus`, `ondraggesture`, etc.) et l'action est décrite dans un langage de script comme JavaScript ;

- la partie `skin` : le style de chaque composant graphique est décrit en utilisant CSS ;

- la partie `locale` : l'internationalisation est réalisé en utilisant le principe des entités XML.

3.1.3. Synthèse

Le nombre de composants graphiques supportés par UIML est potentiellement infini puisqu'il dépend d'un vocabulaire externe. Cette caractéristique est importante car elle permet de décrire des interfaces graphiques pour plusieurs terminaux. XUL offre un vocabulaire abstrait qui couvre l'ensemble des composants graphiques des boîtes à outils existantes sur les stations de travail (Swing, MFC, MAC). Par contre, lorsqu'il s'agit de décrire des interfaces en dehors de ce contexte, XUL n'est pas adapté. En particulier, il ne permet pas de décrire des cartes WML. Au niveau du modèle événementiel, XUL est très basique puisqu'il repose sur le modèle événement-action existant dans HTML. UIML est plus expressif car il permet d'ajouter des conditions avant d'activer les actions.

Au niveau de l'implantation, XUL a été expérimenté en grandeur nature à travers de multiples applications (Mozilla, Chatzilla, etc.). Il existe aussi quelques implantations de UIML. Paradoxalement, c'est UIML qui est le moins complexe à implanter car sa définition est basée sur le langage d'implantation sous-jacent.

Notre choix s'est clairement porté sur UIML pour les raisons évoquées ci-dessus.

3.2. Éditeur réalisé : l'éditeur Docbook

Nous avons mené une expérimentation plus importante sur l'édition de documents Docbook. L'implantation de cette application a été effectuée par deux stagiaires de DESS génie informatique que nous avons encadrés.

L'interface utilisateur de cet éditeur est illustrée figure 53. La fenêtre, les menus, la barre d'outils, ainsi que le contenu du document sont décrits par des éléments UIML. Le menu fichier permet d'activer les fonctions de gestion documentaire, comme l'ouverture, de la fermeture et la sauvegarde du document. Le menu édition offre les fonctions d'édition de haut niveau comme le copier/coller. Le menu insertion permet d'insérer les éléments Docbook. Le menu affichage permet de choisir entre plusieurs présentations du document.

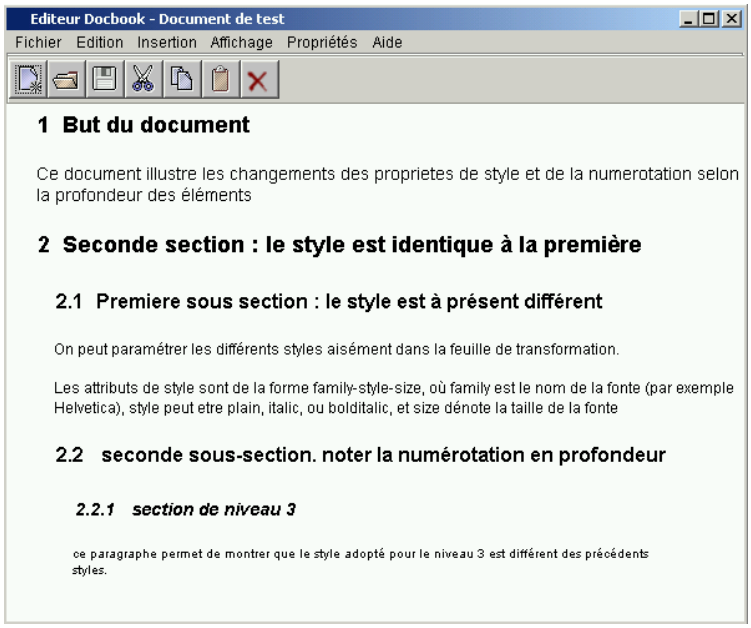


Figure 53. Fenêtre principale de l'éditeur Docbook.

Cet éditeur permet de modifier le contenu des titres et des paragraphes de la même façon que pour l'éditeur SlideShow (cf. section 2.3). Cependant les limites de cet éditeur sont moins importantes que pour l'éditeur SlideShow grâce à une implantation du langage UIML dans notre système d'édition. En particulier, il est possible d'ajouter, entre autres, de nouveaux paragraphes et de nouvelles sections directement dans la vue de présentation en activant la commande correspondante dans le menu insertion.

L'éditeur a été entièrement réalisé en utilisant les langages XSLT et UIML. La spécification complète de l'éditeur représente 480 lignes de code.

3.3. Bilan

Dans cette section, nous avons présenté notre approche pour générer un éditeur dédié à un modèle de document particulier. Nous avons choisi une approche homogène, c'est-à-dire que l'éditeur est généré de la même façon que la présentation, c'est-à-dire par un processus de

transformation. L'avantage de cette solution est que l'éditeur peut être lui-même édité. Les éléments de l'interface utilisateur sont des entités, à un certain niveau équivalentes, aux entités multimédia. L'édition d'un éditeur est similaire à la conception de feuilles de transformation servant à la génération d'une présentation multimédia.

Cet outil est dans la lignée des outils de génération comme Centaur [Borras 88], Thot [Thot], ou plus récemment SmartTools [Attali 01]. SmartTools permet de développer très rapidement des environnements spécialisés pour des langages de programmation ou des langages métiers. Notre démarche est identique, cependant nous utilisons des langages de description et non pas des langages de programmation au cœur de la génération. SmartTools permet d'utiliser le langage XSLT, cependant seul un sous-ensemble de ce langage est permis dans le but de permettre une mise à jour incrémentale.

4. Édition de la transformation

Dans cette section nous décrivons le d'édition à partir des vues cible : l'édition de feuille de transformation.

4.1. Principe général

Du point de vue de l'auteur, l'édition des feuilles de transformation est en partie identique à l'édition d'un document multimédia. Il a la possibilité de manipuler directement les objets du document à partir des différentes vues cible du système d'édition. Par exemple, il peut sélectionner une région dans la vue de présentation et le déplacer de façon progressive. Cependant, à la différence d'un environnement d'édition classique, qui modifie directement le langage de présentation sous-jacent, notre système d'édition opère sur la feuille de transformation pour générer le document de présentation. La traduction de chaque action doit être reconsidérée afin de modifier la spécification de la transformation.

Les actions fournies par les vues ne permettent pas d'éditer une présentation qui dépend du document source. Par exemple, l'auteur veut pouvoir générer une liste de texte pour chaque titre de section. Pour cela, le système d'édition doit être étendu pas des actions permettant de lier des nœuds source à des nœuds cible.

Nous décrivons ces deux types d'édition dans les deux sous-sections suivantes.

4.2. Actions d'édition

Dans cette section, nous considérons plusieurs actions particulières de la part de l'utilisateur. Certaines de ces actions sont issues de l'outil d'édition de document multimédia Madeus [Layaïda 97], et nous serons amené à en définir de nouvelles. Les actions issues de Madeus que nous considérons sont exclusivement réalisées à partir de la vue présentation. Ces actions sont les suivantes :

- le déplacement d'une région. Le déplacement permet d'appliquer une modification sur la valeur des attributs left et top ;
- l'activation de la commande d'ajout d'un groupe spatial et d'une région ;
- l'activation de la commande d'ajout d'un acteur de type particulier. Cette action consiste à ajouter un nouvel acteur ainsi qu'une nouvelle région associée à l'acteur.

Par souci de simplification, nous ne considérons pas les actions dans les autres vues (timeline, attribut, etc.). Les principes que nous énonçons dans la suite peuvent être facilement adaptés dans les autres vues et pour les autres dimensions du document.

4.3. Édition sans lien avec le document source

Pour déplacer une région, l'auteur peut sélectionner un objet dans la vue présentation. Supposons qu'une règle de transformation consiste en la génération d'une région pour chaque transparent du document source. Dans ce cas, lorsque l'auteur sélectionne une des régions générées puis la déplace, une traduction naturelle est de modifier directement l'instruction qui a servi à produire cette région. Cette modification entraîne des répercutions sur les autres régions générées par cette instruction qui sont aussi déplacées. Une autre solution est de ne déplacer uniquement que la région sélectionnée. Cet exemple permet d'un premier abord d'identifier deux modes d'édition : le mode global et le mode local. Dans le mode global, la modification est appliquée à l'ensemble des objets générés par la même règle de transformation. Dans le mode local, la modification intervient uniquement sur l'objet sélectionné par l'auteur.

Dans la suite, nous décrivons ces deux modes d'édition. Dans la seconde sous-section nous proposons une généralisation des deux modes. Une difficulté de la traduction est la prise en compte des liens inter-dimentionnels que nous étudions dans la section 4.3.3. Nous terminons par la réalisation de ce type d'édition.

4.3.1. Édition globale et locale

Dans le cas d'une opération d'édition ayant une portée globale, le générateur modifie la feuille de transformation de telle façon à appliquer la modification sur l'ensemble des nœuds cible générés par la même instruction XSLT que l'élément sélectionné. Par exemple, prenons la spécification suivante :

```
1. <xsl:template match="slides">
2.     <madeus:flow>
3.         <madeus:region id="titre"/>
4.         <xsl:for-each select="slide">
5.             <madeus:region>
6.             …
7.             </madeus:region>
8.         </xsl:for-each>
9.     </madeus :flow>
10.    </xsl:template>
```

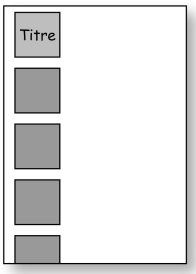


Figure 54. Résultat graphique de l'exécution de la transformation

Cette spécification génère un élément flow qui organise spatialement les régions qu'il contient de haut en bas (par défaut). Ensuite, pour chaque élément slide du document source, une région, représentée par un carré gris dans la figure 54, est générée.

Lorsque l'auteur déplace la quatrième région dans le mode global, le générateur produit le résultat suivant :

```
1. <xsl:template match="slides">
2.     <madeus:flow>
3.         <madeus:region id="titre"/>
4.         <xsl:for-each select="slide">
5.             <madeus:region left="33">
6.             …
7.             </madeus:region>
8.         </xsl:for-each>
9.     </madeus :flow>
10.    </xsl:template>
```

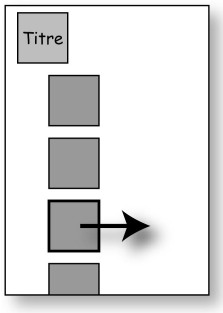


Figure 55. Représentation graphique d'une modification globale.

Dans ce mode de génération, l'opération d'édition consiste en une modification directe de la feuille de transformation. L'attribut left est ajouté sur la région qui est générée pour chaque élément slide.

Lorsque l'auteur effectue une modification sur une région dans le mode local, cette modification est appliquée uniquement à la région sélectionnée. Pour cela, la règle de transformation qui a généré la région est recherchée selon le processus de transformation inverse décrit dans la section 2.2.2. Une fois la règle trouvée, la difficulté est d'identifier l'exécution de cette règle qui a effectivement généré la région modifiée. En effet, dans notre exemple, une région est générée pour tous les éléments source de type slide. Par exemple lorsque l'auteur déplace la quatrième région seule cette région doit être déplacée. Le résultat de l'action peut se traduire par la spécification suivante :

```
1. <xsl:template match="slides">
2.     <madeus:flow>
3.         <madeus:region id="titre"/>
4.         <xsl:for-each select="slide">
5.             <madeus:region>
6.                 <xsl:if test="position()=3">
7.                     <xsl:attribute name="left">33
8.                     </xsl:attribute>
9.                 </xsl:if>
10.                …
11.                </madeus:region>
12.            </xsl:for-each>
13.        </madeus :flow>
14.    </xsl:template>
```

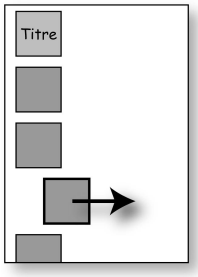


Figure 56. Résultat après le déplacement de la 4ième région.

La modification de la transformation se résume par l'ajout d'un test qui isole la région. Lorsque la condition du test est vérifiée, alors la transformation génère un attribut left dont la valeur correspond à la position actuelle de la région. Dans cet exemple, la condition est calculée en fonction de la position de la région dans le document cible. Dans la feuille de

transformation cette région est générée pour le troisième transparent (la première région ligne 3 ne dépend pas des transparents source), d'où le test position()=3.

4.3.2. Portée de l'édition à un ensemble d'objets

Les deux modes de génération que nous avons présentés peuvent être généralisés en considérant plusieurs conditions d'identification de l'objet à modifier. Dans le mode local, la condition repose sur la position de l'élément dans l'arbre cible, tandis que dans le mode global, aucune condition n'est posée. Ces deux cas ne sont pas les seuls. Par exemple, l'auteur peut vouloir déplacer les objets qui ont une position impaire. Toutefois, nous limitons la généralisation aux conditions qui portent sur la position du nœud cible dans le document cible.

4.3.3. Liens inter-dimensionnels

Dans le contexte de l'édition de documents multimédias, le résultat d'une action se traduit souvent par l'ajout de liens entre les différentes structures du document. Par exemple, reprenons la première spécification de la section 4.3.1, que nous rappelons ci-dessous :

```
1. <xsl:template match="slides" mode="spatial">
2.     <madeus:flow>
3.         <madeus:region id="titre"/>
4.         <xsl:for-each select="slide">
5.             <madeus:region/>
6.         </xsl:for-each>
7.     </madeus:flow>
8. </xsl:template>
```

Supposons que l'auteur sélectionne les trois premières régions et qu'il active la commande d'ajout d'un acteur de type image. Le résultat des opérations d'édition conduit à la spécification suivante :

```
1. <xsl:template match="slides" mode="actor">
2.     <xsl:for-each select="slide">
3.         <xsl:if test="position() <= 3">
4.             <madeus:image id="A-Image-l1-{generate-id()}"
5.         </xsl:if>
6.     </xsl:for-each>
7. </xsl :template>
8.
9. <xsl:template match="slides" mode="spatial">
10.        <madeus:flow>
11.            <madeus:region id="titre"/>
12.            <xsl:for-each select="slide">
13.                <madeus:s-group>
14.                    <xsl:if test="position() <= 3">
15.                        <madeus:region actor="A-Image-l1-{generate-id()}"/>
16.                    </xsl:if>
17.                </madeus:s-group>
18.            </xsl:for-each>
19.        </madeus:flow>
20.    </xsl:template>
```

Une région a été ajoutée pour contenir les nouveaux acteurs, ce qui correspond aux lignes 14 à 16. De plus, une règle entière a été ajoutée pour générer les acteurs. Cette nouvelle règle est instanciée pour les mêmes éléments que la règle qui génère la structure spatiale (d'où le mode `spatial`) mais dans un mode différent (le mode `actor`). Le contenu de la nouvelle règle reprend l'ensemble des instructions XSLT de la règle spatiale.

Dans Madeus 2.0, les liens inter-dimensionnels sont représentés par des identificateurs (cf. chapitre 4 section 3.2). Ces identificateurs sont obligatoirement générés dynamiquement. Pour cela, la fonction `generate-id()` est utilisée : elle génère un identificateur unique pour chaque nœud source, cet identificateur étant toujours le même pour un nœud source donné. Cependant cette fonction ne suffit pas à générer un identificateur unique car pour un même nœud source, plusieurs liens peuvent être définis. Par conséquent, à chaque identificateur est ajouté un nom correspondant au nom statique du lien. Dans l'exemple, ce nom est `l1`.

Il faut noter que ces liens inter-dimentionnels existent entre les éléments Madeus 2.0 et aussi entre les instructions XSLT. Par exemple, il existe un lien entre les deux instructions `for-each` lignes 2 et 12. En effet, la suppression ou la modification de l'expression d'une instruction `for-each` entraîne obligatoirement la modification de l'autre instruction `for-each`.

4.3.4. Réalisation

Pour résumer, la modification de la feuille de transformation repose sur les paramètres suivants :

- le ou les éléments cible sélectionnés par l'auteur. De ces éléments sont inférées zéro ou plusieurs conditions sur la position. Le mécanisme d'inférence repose sur la reconnaissance de schémas de sélection prédéfinis. Par exemple, si l'auteur sélectionne le premier et le troisième élément d'un groupe, alors le moteur d'inférence calcule et propose à l'auteur la condition suivante exprimée dans le langage XPath : `not(position() mod 2 = 0)` ;

- l'action effectuée par l'auteur accompagnée de la valeur des paramètres associés. Par exemple, l'action d'ajout d'un acteur est accompagnée du type de média de l'acteur ;

- la feuille de transformation courante.

À partir de ces paramètres, la modification de la feuille de transformation s'effectue selon le pseudo-algorithme suivant que nous illustrons à chaque étape avec l'exemple d'une action e déplacement de deux régions (cf. figure 57) :

1. à partir du premier paramètre, c'est-à-dire les nœuds sélectionnés dans le document cible, le générateur identifie les instructions de la feuille de transformation qui correspondent aux nœuds cible. Cette identification est effectuée de la même façon que dans la première étape de la transformation inverse (cf. section 2.2.2). Cette étape permet de déterminer les nœuds d'exécution de la transformation (cf. section 2.2.2) qui ont généré les nœuds cible ainsi que leur position dans le nœud parent de l'arbre d'exécution (l'instruction `for-each` de la figure 57) ;

2. les instructions XSLT qui ont généré les nœuds de l'arbre d'exécution sont identifiées. La position des nœuds d'exécution est convertie en position du nœud source pour lequel les nœuds d'exécution ont été générés. Par exemple, les deux premières régions de

l'arbre d'exécution ont été générées par les deux premiers nœuds source sélectionnés par l'instruction for-each. Dans ce cas précis, les deux positions sont identiques ;

3. à partir de ces positions, une condition est formée. Si cette condition n'est pas toujours évaluée à vraie, c'est-à-dire si plusieurs nœuds cible peuvent être générés à partir de l'instruction, alors cette instruction est isolée des autres exécutions. La création de la condition et la génération des instructions pour isoler les exécutions dépendent de la feuille de transformation courante. Dans l'exemple, la condition créée est l'expression position()=1 or position()=2, et l'instruction if est ajoutée pour isoler les deux régions ;
4. le code correspondant à l'opération d'édition est modifié ou créé. Dans ce cas précis, l'instruction attribute est créée avec comme nom d'attribut left et comme valeur 33.

Seule cette dernière étape dépend des actions fournies par la vue. Les autres étapes doivent être appliquées quelle que soit l'action utilisateur.

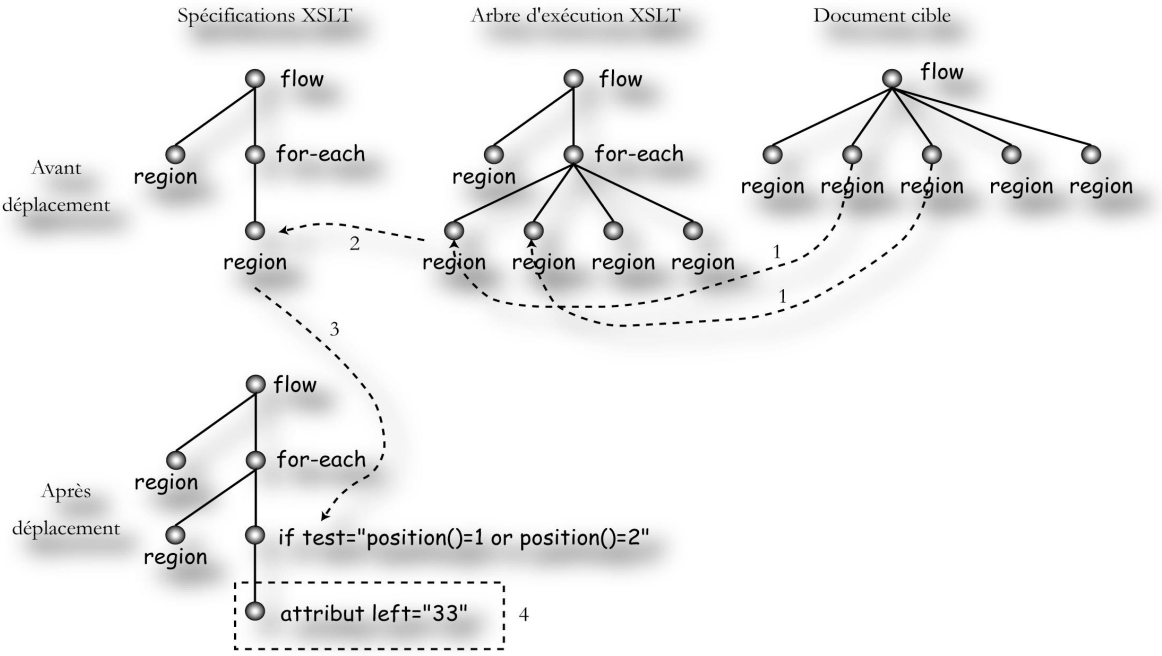


Figure 57. Etapes pour générer le déplacement de deux régions.

Nous ne détaillons par la réalisation de la troisième étape. Cependant, il est possible de donner quelques pistes pour l'implanter :

- la porté des instructions à considérer lors de la création de la condition et la génération du code d'isolation est limitée vers le haut par une instruction (for-each ou apply-templates) qui exécute plusieurs fois les instructions qu'elle contient. En effet, les positions calculées lors de la troisième étape sont relatives à cette instruction ;
- les instructions utilisées pour implanter l'isolation sont restreintes aux instructions if et choose. Il suffit donc d'énumérer les combinaisons possibles entre ces instructions pour déterminer quelles instructions générer.

L'implantation se résume donc à une étude de cas sur un nombre restreint d'instructions. Nous avons donné un cas particulier lorsque la région est un fils de l'instruction for-each et aucune autre condition n'existe. Cependant nous avons posé les bases fondamentales sur lesquelles cette étude de cas peut être généralisée.

4.4. Édition avec lien avec le document source

Pour éditer la transformation en fonction des informations du document source, l'auteur doit être en mesure d'extraire un sous-ensemble de ces informations puis d'indiquer un endroit dans une vue cible où le placer. La première étape est accomplie grâce à la création d'une expression XPath. Nous commençons par rappeler la syntaxe d'une telle expression puis nous montrons comment en créer de façon interactive. Nous verrons ensuite les règles de génération de transformations après que l'auteur ait lié une expression XPath avec le document cible.

4.4.1. Le langage XPath

Nous introduisons le vocabulaire de base d'une expression XPath à travers l'exemple figure 58 Cet exemple est une expression de type particulier appelée *sélecteur*. Un sélecteur désigne un ou plusieurs nœuds. Il est constitué d'une succession d'*étapes* séparées par le caractère '/'. Chaque étape est composée d'un *axe*, d'un *test de nœud* et d'un ou plusieurs *prédicats* :

- l'axe spécifie le type de relation arborescente entre le nœud contextuel et les nœuds à localiser ;
- le test de nœud spécifie le type du nœud et le nom des nœuds obtenus par l'étape de localisation ;
- zéro ou plusieurs prédicats, qui sont des expressions booléennes pour raffiner l'ensemble des noeuds obtenus par l'étape de localisation.

Le sélecteur de la figure 58 prend tous les nœuds (*) qui sont descendant (axe descendant) de la section qui est à la position 2. Les sections sont sélectionnées à partir du nœud source contextuel lors de l'exécution de la transformation (cf. section 2.2.2).

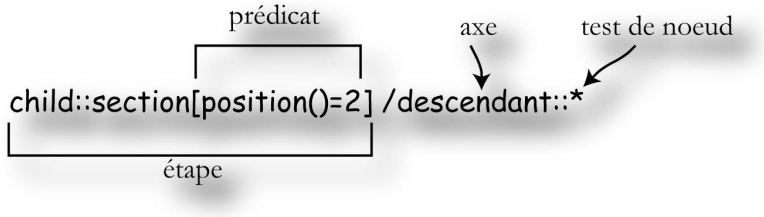


Figure 58. Exemple d'une expression XPath de type sélecteur.

Lorsqu'un sélecteur commence par le caractère '/', il sélectionne le nœud racine du document.

Les expressions sont utilisées de façon plus large que la sélection de nœuds. Notamment, il est possible d'écrire des expressions numériques, booléennes ou manipulant des chaînes de caractères. Par exemple, l'expression count(section) >= 10 teste si le nombre de sections est supérieur ou égal à dix.

4.4.2. Édition d'une expression XPath

L'édition d'une expression XPath s'effectue directement dans la vue hiérarchique du document source (cf. section 2.1). Pour cela, nous avons étendu cette vue afin d'éditer une expression. Cette extension consiste à visualiser l'expression en cours d'édition et de montrer le résultat de l'évaluation de l'expression.

Les opérations d'édition ajoutées à cette vue sont les suivantes :

- ajout/suppression d'une étape dans l'expression : cette opération est effectuée en sélectionnant/désélectionnant simplement le nœud. L'ordre de sélection étant important dans cette opération, il est donc affiché ;
- ajout/suppression/modification d'un prédicat pour une étape.

À partir de ces informations, le système d'édition est capable de générer une expression en déduisant les axes. Le calcul des axes dépend de la différence de profondeur de deux nœuds n_1 et n_2 sélectionnés successivement :

- le niveau de profondeur de n_1 et n_2 sont identiques. Deux cas peuvent se produire :
  - si ces deux nœuds ont le même nœud ascendant direct, alors ces nœuds sont voisins (sibling). Par conséquent, si n_2 est après le nœud n_1, le système auteur génère par défaut un axe de type following-sibling. Dans le cas contraire, si n_2 est avant le nœud n_1, l'axe de type preceding-sibling est généré ;
  - sinon les axes following ou preceding sont générés selon que le nœud n_2 soit après ou avant n_1 ;
- la différence de niveau entre n_1 et n_2 est égale à un. Deux cas peuvent se produire :
  - si n_1 et n_2 partagent les mêmes nœuds ancêtre, alors les axes child ou parent sont générés selon l'ordre de sélection de n_1 et n_2 ;
  - sinon les axes following ou preceding sont générés selon que le nœud n_2 soit après ou avant n_1 ;
- la différence de niveau en n_1 et n_2 est supérieure à un. Ce cas est similaire au précédent, sauf que les axes child et parent sont remplacés respectivement par les axes descendant et anscestor.

Pour éditer les prédicats et l'union d'expressions, le système d'édition repose sur le principe d'empilement. L'expression en cours d'édition est stockée, l'auteur ayant alors la possibilité d'éditer une nouvelle expression dans la vue.

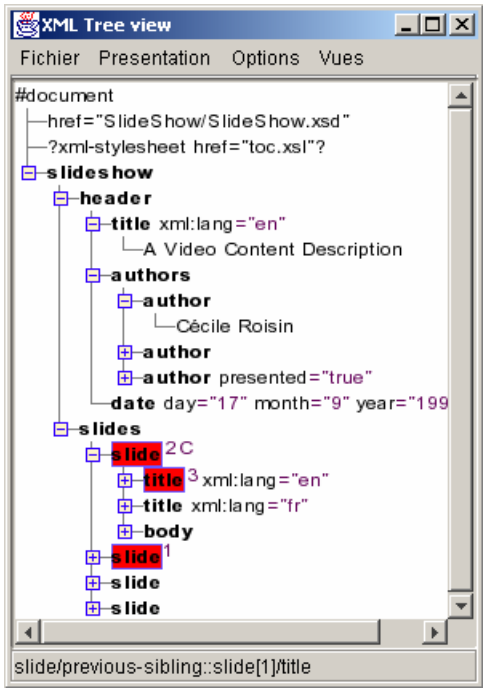


Figure 59. Édition de l'expression slide/previous-sibling::slide[1]/title.

Grâce à cette vue, l'auteur peut éditer ou modifier simplement une expression quelconque. Il peut aussi vérifier le résultat de l'évaluation de son expression en sélectionnant le nœud source à partir duquel l'expression est évaluée. Il faut noter cependant que les expressions pouvant être éditées par cette vue dépendent fortement du document source. L'expression de la figure 59 ne peut pas être éditée si le document ne contient qu'un seul élément slide.

4.4.3. Identification des paramètres de génération

Après avoir édité une expression, l'auteur la connecte avec le document de présentation en utilisant une des deux méthodes ci-dessous :

- en déplacement et en lâchant l'expression dans une des vues cible. Cette méthode permet d'identifier un seul nœud cible. La position du lâché est un paramètre du générateur.
- en sélectionnant un ou plusieurs nœuds cible et en actionnant la commande de connexion. Par rapport à l'action précédente, aucune position n'est passée au générateur.

Quelle que soit la méthode utilisée, le résultat est l'identification *de nœuds cible*. Par conséquent, au niveau de la feuille de transformation, seules des instructions de génération peuvent être sélectionnées. Ces instructions sont caractérisées par le *type* de nœuds qu'elles génèrent (élément, attribut, etc.), par le *nom* de ces nœuds, par le *contexte de déclaration* et par le *contexte d'exécution*.

Les caractéristiques d'une expression XPath sont les suivantes :

- *le type du résultat de l'évaluation*. Ce type est un booléen, une chaîne de caractères, un nombre ou un ensemble de nœuds. Il faut noter que le type du résultat dépend de l'instruction associée à l'expression. Par exemple, l'expression slides/slide sélectionne

un ensemble de nœuds qui est converti en un booléen lorsque l'expression est associée à l'instruction **if**. L'instruction **apply-templates** utilise cet ensemble de nœuds sans le convertir vers un autre type. Par conséquent, il n'est pas possible de connaître le type du résultat avant la génération. Par contre, certains types ne peuvent pas être convertis vers d'autres types. Par exemple, un booléen ne peut pas être converti vers un ensemble de nœuds. Cela permet de réduire le nombre de cas de génération ;

- lorsque le résultat est un ensemble de nœuds, le *nombre* et le *type* des nœuds (élément, attribut, etc.) de cet ensemble font partie de ces caractéristiques.

Le nombre de paramètres de génération s'élève à huit. Le résultat de la génération dépend de la valeur de ces paramètres. Chaque valeur différente peut entraîner un résultat de génération différent. Lorsque la valeur des paramètres est de nature discrète, alors il est nécessaire d'effectuer une étude de cas pour chacune de ces valeurs. En combinant les paramètres discrets deux à deux et en ne considérant qu'un sous-ensemble des types de nœuds, l'estimation du nombre de résultats possibles est supérieur à 100. L'objectif de ce chapitre n'est pas de décrire de façon détaillée l'ensemble de ces cas de génération. Nous nous restreignons aux expressions qui sélectionnent un ensemble de nœuds et nous ne considérons que deux paramètres : le nombre de nœuds sélectionnés et le type des nœuds cible.

4.4.4. Quelques cas de génération

Nous distinguons trois paliers pour le nombre de nœuds sélectionnés par l'expression XPath : aucun, un seul et plusieurs :

- lorsque aucun nœud source n'est sélectionné, une erreur est affichée à l'auteur ;
- lorsqu'un seul nœud source est sélectionné, plusieurs cas peuvent se produire selon le type du nœud cible :
    - le nœud cible est la valeur d'un attribut. Le générateur ajoute un attribut dont la valeur est l'évaluation de l'expression. Par exemple :

```
<Image FileName=""/>
```

(a) Avant

```
<Image FileName="{photo[1]/@src}"/>
```

(b) Après

    - le nœud cible est le contenu d'un élément. Le générateur ajoute un nouvel acteur dans l'élément cible ou modifie un de ses attributs. Le type du nouvel acteur dépend de la nature de l'élément cible et du nœud source. Par exemple, lorsque la valeur du nœud source référence un fichier image et lorsque l'élément cible est un groupe spatial, le générateur ajoute une région dans ce groupe et ajoute aussi un nouvel acteur de type image ;
    - le nœud cible est un attribut ou un élément. Le générateur ajoute un test décrit par l'instruction **if**, dont la condition est basée sur l'expression ;
- lorsque plusieurs nœuds source sont sélectionnés, les cas suivants peuvent se produire :
    - le nœud cible est la valeur d'un attribut. Une instruction **for-each** est générée. L'attribut associé **select** contient le fragment de l'expression qui génère plusieurs nœuds. Le contenu de l'instruction **for-each** est un attribut. Par

exemple, la génération suivante est produite lorsque l'auteur lâche l'expression photo/@src dans le champ FileName de la vue attribut :

```
<xsl:template match="album" mode="ac">
        <Image ID="A-1"/>
</xsl:template>

<xsl:template match="album" mode="sp">
        <S-Group>
            <Region Actor="A-1"/>
        </S-Group>
</xsl:template>
```

(a) Avant

```
<xsl:template match="album" mode="ac">
        <xsl:for-each select="photo">
            <Image ID="A-1{generate-id()}"
                FileName="{@src}"/>
        </xsl:for-each>
</xsl:template>

<xsl:template match="album" mode="sp">
        <S-Group>
          <xsl:for-each select="photo">
            <Region
                Actor="A-1{generate-id()}"/>
          </xsl:for-each>
        </S-Group>
</xsl:template>
```

(b) Après l'action lâcher

- le nœud cible est le contenu d'un élément. Un message d'erreur est affiché puisque XML ne permet qu'un seul attribut de même nom sur un élément ;
- le nœud cible est un attribut ou un élément. Lorsque l'élément cible est un groupe, le générateur produit plusieurs acteurs. Par contre, lorsque l'élément est une feuille, alors une erreur est affichée pour les mêmes raisons que dans le cas précédent.

4.5. Bilan

Dans cette section, nous avons identifié les principales étapes pour éditer des feuilles de transformation selon une approche par manipulation directe. Pour chacune de ces étapes nous avons proposé un ensemble d'actions utilisateur à travers une interface graphique et plusieurs algorithmes pour modifier la transformation pour chacune de ces actions. Nous avons restreint notre étude à la vue d'exécution et à un sous-ensemble des paramètres de génération identifiés dans la section 4.4.3. Cependant, notre approche peut facilement être généralisée à l'ensemble des vues cible, en particulier la vue timeline et la vue attribut, et à l'ensemble des paramètres de génération.

5. Synthèse sur le développement

Avant d'effectuer une synthèse sur les différentes tâches de développement réalisées tout au long de cette thèse, il est nécessaire de revenir sur l'historique des travaux sur les documents multimédias accomplis au sein du Projet Opéra. Durant la thèse de Nabil Layaida [Layaïda 97] et celle de Loay Sabry [Sabry 99] a été développé le premier prototype Madeus (version 0.0). Le terme Madeus désigne à la fois ce prototype d'édition et de présentation de documents multimédias et le langage de description de scénarios multimédias utilisé dans ce prototype. Ce

prototype, initialement implanté en utilisant le langage C, a été porté en Java au début de la thèse de Laurent Tardif [Tardif 00]. Est apparu alors Madeus 1.0 reprenant l'esprit de Madeus 0.0 tout en offrant de nouvelles fonctionnalités pour augmenter la productivité et la convivialité de l'édition de documents multimédias notamment grâce à la vue temporelle. Madeus 1.0 est une boîte à outils facilitant le développement d'un environnement d'édition pour un langage multimédia donné, en particulier Madeus et SMIL 1.0. Le modèle de document interne de Madeus 1.0 est un modèle pivot permettant de représenter l'ensemble des modèles de documents multimédias.

Les travaux de cette thèse ont contribué de façon importante pour faire évoluer ce prototype en abandonnant l'approche boîte à outils et en intégrant l'esprit XML à travers la spécialisation de l'interface de programmation DOM pour chacun de modèles de documents éditables. De plus, un des objectifs a été d'étendre cet outil par un processus de transformation. La réalisation de cette autre évolution de Madeus, dite 2.0, s'est effectuée grâce à la participation de Tien Tran_Thuong concernant l'édition spécifique de documents Madeus 2.0 avec comme particularité d'éditer la structure et les méta-informations de la vidéo [Roisin 99b, TTran 01]. Dans le cadre de cette thèse, les évolutions portent principalement sur les points suivants :

- le développement d'une partie du modèle de document Madeus 2.0 décrit dans la section 3.2 du chapitre précédent et du modèle UIML (cf. section 3.1.1 de ce chapitre). L'implantation de ce modèle concerne d'une part l'implantation quasi-complète du standard DOM level 2 concernant les modules *core*, *event* et *traversal* pour représenter de façon interne un document Madeus 2.0 et UIML. D'autre part, le formatage de Madeus 2.0 implanté dans le cadre de cette thèse concerne les modules acteur et spatial. Le formatage des éléments UIML a été effectué par deux stagiaires de DESS encadrés également dans le cadre de cette thèse ;

- le développement de la vue d'exécution, que ce soit pour la présentation du document multimédia et pour l'édition. La présentation concerne le développement d'un ensemble de « players » afin de restituer l'information à travers plusieurs médias, en particulier le texte simple, le texte HTML, les images JPG, GIF et SVG, le son, la vidéo, etc. Plusieurs actions d'édition ont été implantées, comme l'insertion, le déplacement de médias et le déplacer-lâcher, dont les effets sont de modifier soit le document source, soit la feuille de transformation selon le mode choisi par l'auteur ;

- le développement de la vue hiérarchique telle qu'elle est décrite dans la section 2.1 et la section 4.4.1. Cette vue permet de visualiser un document XML sous forme hiérarchique et d'éditer les éléments et les attributs de ce document, soit directement dans la vue, soit à travers un menu contextuel. La construction de ce menu s'appuie sur la description formelle en XML-Schema du modèle du document édité. Un sous-ensemble du langage XML-Schema est reconnu, à savoir les élements element et sequence, ainsi que les attributs associé name et ref.

- l'intégration d'un processeur de transformation XSLT permettant ainsi de construire entièrement l'architecture telle qu'elle est décrite dans la section 2.2 du chapitre précédent. De plus, cette intégration, couplée au développement d'une partie de la transformation inverse, permet d'éditer des documents XML à partir d'une de ses représentations (cf. section 2 de ce chapitre). Nous verrons dans le chapitre suivant

comment nous avons modifié ce processeur de transformation XSLT pour le rendre incrémental.

Pour conclure, Madeus 3.0 a été écrit presque entièrement en Java et représente actuellement 60000 lignes de code (sans commentaire). Environ 30000 lignes ont été développées dans le cadre de cette thèse concernant uniquement Madeus 2.0 (cf. chapitre suivant pour le processeur de transformation incrémental). L'utilisation de l'environnement Java explique la quantité relativement importante du code réalisé. En effet, les nombreuses fonctions offertes par cet environnement, ainsi qu'une documentation complète, l'accès aux sources et la possibilité de re-architecturer automatiquement l'organisation du code sont autant de critères qui permettent de gagner du temps de développement.

6. Conclusion

Dans ce chapitre nous avons décrit notre outil d'édition qui permet de manipuler des documents XML de n'importe quel type tout en offrant une interface utilisateur conviviale et efficace. L'auteur modifie son document XML à travers une de ses présentations, soit en modifiant le document de présentation, soit en agissant indirectement sur le document source. De plus, nous avons présenté une première expérience concernant l'édition de transformations par manipulation directe. Nous avons proposé des solutions à la fois au niveau de l'interface graphique et aussi au niveau de la traduction des actions de l'utilisateur en opération d'édition sur la transformation. Grâce à cet outil, il est possible d'éditer de façon interactive des documents adaptables. Enfin, les principes d'édition de transformations par manipulation directe décrits dans la section 4 ont fait l'objet d'un article présenté lors de la conférence DocEng'01 [Villard 01d].

L'édition de ces deux ensembles d'informations est le premier pas vers une édition interactive de présentations adaptées au contexte de l'utilisateur. Le problème d'éditer de multiples présentations correspondant à différentes adaptations est complexe car, compte tenu du nombre potentiellement élevé des contextes utilisateur possibles, il est impossible de naviguer dans l'espace des solutions d'adaptation.

L'utilisation effective de notre outil d'édition met en évidence le manque de réactivité de la part du système après une action utilisateur. Cependant, un des composants qui est au cœur du système d'édition que nous proposons reste à définir. Il s'agit du composant permettant la transformation incrémentale. Il fait l'objet du chapitre suivant.

Chapitre 6

Transformation incrémentale

Tout au long de ce mémoire, nous avons fait l'hypothèse de l'existence d'un processeur de transformation incrémental. L'objectif de ce chapitre est de proposer plusieurs solutions techniques pour réaliser un tel processeur de transformation. Pour étudier ces solutions, il est indispensable de spécifier avec précision et sans ambiguïté la sémantique du langage XSLT. C'est pourquoi nous avons choisi une approche formelle. Nous serons alors en mesure d'analyser les conséquences d'une modification du document source et de la spécification de la transformation sur le document cible, pour ensuite proposer des solutions en accord vis-à-vis de ces conséquences. Nous confrontons ces solutions qui fondent notre processeur de transformation incrémental avec un processeur classique.

1. Introduction

1.1. Objectifs

L'objectif du processeur de transformation incrémental est de mettre à jour le document résultat de la transformation après une modification soit du document source, soit de la spécification de la transformation. Pour que cette mise à jour soit efficace, le processeur incrémental ne doit réexécuter que les fragments de la transformation qui produisent un résultat différent. L'origine d'une mise à jour peut être très variée puisqu'elle peut consister au changement de la valeur d'un attribut ou à l'ajout et/ou la suppression de fragments d'arbre. La difficulté est d'identifier ces fragments en fonction du type de modification appliquée sur le document source ou sur la spécification de la transformation.

Lorsque le document source est modifié, la difficulté est principalement située au niveau du langage de sélection de nœuds source et au niveau de l'usage de ce langage. Pour le langage CSS, ou plus généralement pour les langages de transformation dirigés par la source (cf. section 3.2.1.1 du chapitre 2), le langage de sélection s'appuie uniquement sur le type de 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. Il n'y a donc pas de problèmes particuliers pour réaliser la mise à jour incrémentale [Amaya].

Par contre, le langage XSLT va plus loin sur ces deux aspects. Le langage de sélection est plus expressif puisqu'il permet notamment d'exprimer des sélections en fonction du contexte d'exécution au moment de son évaluation. De plus, le langage de sélection peut être utilisé lors de la génération du document cible. Enfin, les règles en XSLT doivent être exécutées dans un ordre précis et dans un contexte donné qui est constamment remis à jour pendant l'exécution de la transformation. Toutes ces caractéristiques de XSLT sont autant de difficultés pour réaliser l'opération de mise à jour incrémentale. L'objectif de ce chapitre est de montrer qu'il existe cependant des solutions pour une partie des transformations.

1.2. Approches existantes

Puisque XSLT s'apparente en partie à un langage de programmation, nous avons étudié les approches incrémentales dans ce domaine. Par analogie, la transformation incrémentale est équivalente à une exécution incrémentale d'un programme. Dans ce contexte, plusieurs travaux portent sur l'exécution incrémentale de programmes impératifs [Agrawal 91, Netzer 94]. Généralement le but de ces travaux est d'offrir des fonctionnalités pour rebrousser chemin (backtracking) lors de la phase de mise au point d'un programme (debug). Pendant cette phase, le programmeur peut annuler l'exécution d'une instruction, changer une donnée initiale du programme et réexécuter le programme de façon incrémentale. Dans ce contexte, l'exécution incrémentale repose sur l'observation de l'exécution du programme pour produire un historique d'exécution. Cet historique est composé d'une suite de couples (ligne, valeurs de variables). À partir de cet historique, l'état de la mémoire peut être restauré et l'exécution peut continuer. Le principal problème des systèmes reposant sur ces travaux est le coût nécessaire pour stocker l'historique et le temps requis pour restaurer le contexte.

Dans le cadre de la transformation incrémentale, le problème se pose différemment si l'on considère les points suivants :

- généralement, l'exécution d'une transformation est limitée, par conséquent la taille de l'historique est nettement inférieure ;
- de plus, l'historique d'exécution ne nécessite pas d'être stocké dans sa globalité. Puisque XSLT est un langage sans effet de bord, restaurer le contexte d'exécution comme la valeur des variables peut être facilement et rapidement effectué. En effet, pour un langage avec effet de bord, le contexte d'exécution peut être potentiellement modifié par l'ensemble des instructions exécutées avant l'instruction à réexécuter (cf. figure 60). Ce n'est pas le cas pour un langage sans effet de bord : seules les instructions ascendantes par rapport à l'instruction à réexécuter peuvent modifier le contexte d'exécution ;

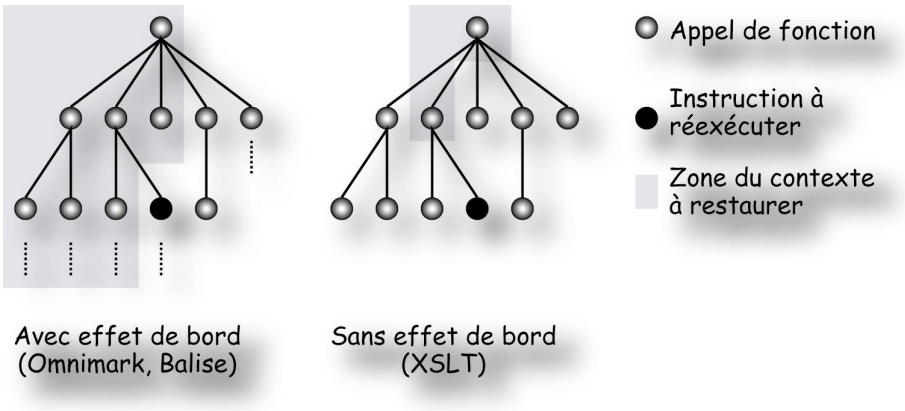


Figure 60. Restauration du contexte d'exécution pour un langage avec effet de bord et sans effet de bord

- la transformation incrémentale s'effectue après qu'une première transformation complète ait été effectuée. Cette propriété est fondamentale dans le cadre éditorial. En effet, lors de l'édition, le document transformé est généralement manipulé dans sa

globalité, et donc l'existence d'une transformation préalable portant sur tout le document est nécessaire ;

- la modification de la spécification de la transformation (le programme) pendant une session incrémentale est possible. Par conséquent, l'édition des feuilles de transformation est possible.

Ces différences sont suffisamment importantes pour laisser espérer qu'un processeur de transformation incrémental puisse être performant et utilisable dans un système interactif. Nous verrons cependant que le principe de notre processeur incrémental est similaire aux systèmes incrémentaux de programmation, notamment sur le fait qu'il s'appuie sur un historique représenté par un arbre qui correspond au flot d'exécution de la transformation (cf. section 2.2.2).

La suite de ce chapitre est organisée de la façon suivante. Nous commençons par décrire le langage XSLT du point de vue du programmeur. Notamment, nous proposons une modélisation formelle basée sur la sémantique dénotationnelle du langage de sélection qui le compose (XPath). À partir de cette formalisation, nous serons capable de proposer nos solutions pour réaliser un processeur de transformation XSLT incrémental. Ces solutions sont étudiées dans la section 2.4. Une évaluation de ces solutions est ensuite effectuée à partir du prototypage que nous avons effectué avec le processeur XSLT Xalan. Ce chapitre se termine par la description de l'utilisation de la transformation incrémentale dans notre outil d'édition.

2. Le langage XSLT

Dans la section 3.2.1.2 du chapitre 2, nous avons décrit le langage XSLT d'un point de vue utilisateur. L'objectif de cette section est de présenter ce langage du point de vue du programmeur d'un processeur XSLT. Pour cela, nous commençons par décrire son fonctionnement interne, c'est-à-dire comment est interprétée une spécification XSLT. Nous décrivons chacune des étapes et les structures de données utilisées. Un des composants clés du processeur incrémental est la détection du changement du résultat de l'évaluation d'une expression XPath. C'est pourquoi nous proposons ensuite une formalisation du langage d'expression XPath au moyen de la sémantique dénotationnelle. Pour les lecteurs non familiers avec cette représentation nous proposons une introduction dans la sous-section suivante.

2.1. Préliminaires

Nous définissons tout d'abord formellement la notion d'arbre originellement introduite par Pair et Quere [Pair 68] et Takahashi [Takahashi 75]. Puis nous décrivons les notions de base de la sémantique dénotationnelle.

2.1.1. Forêt et arbre

La définition d'une forêt est la suivante :

Définition 1 (forêt) : Une forêt dans l'alphabet Σ est :
(1) ε (la forêt vide) ;
(2) $n\langle f\rangle$, avec n qui est un symbole de l'alphabet Σ et f est une forêt ;
(3) fg, avec f et g des forêts.

Un arbre est une forêt, comme l'énonce la définition suivante :

> **Définition 2 (arbre)** : un arbre est une forêt de la forme $n\langle f \rangle$. L'ensemble des arbres couvrant l'alphabet Σ est noté T_Σ.

Dans le contexte XML, un alphabet correspond à l'ensemble des noms d'élément et d'attribut définis dans un modèle de document (DTD). Dans la suite, nous serons amené à considérer deux arbres, l'arbre source et l'arbre cible, couvrant chacun un alphabet différent noté respectivement Σ et Δ.

2.1.2. Sémantique dénotationnelle

La sémantique dénotationnelle est une méthode formelle pour définir la sémantique d'un langage [Allison 87]. Elle repose sur les fondements mathématiques bien connus de la théorie des ensembles. L'objectif de cette section est d'introduire les concepts de base à travers un exemple.

La base de la sémantique dénotationnelle est la *syntaxe* du langage de programmation. Généralement, cette syntaxe est abstraite afin d'avoir une définition plus concise du langage qu'elle modélise. Par exemple, voici la syntaxe abstraite pour les expressions en utilisant la notation BNF :

$$i : Number$$
$$e : Expr \qquad e ::= e + e \mid e - e \mid i$$

$Number$ représente un domaine de valeurs correspondant à l'ensemble des entiers. $Expr$ représente un objet syntaxique correspondant dans ce cas particulier à une expression. Comme le domaine $Number$ est bien connu, sa syntaxe et sa sémantique ne sont pas différenciées. La sémantique des expressions est donnée par une ou plusieurs fonctions sémantiques correspondant chacune à une interprétation. La définition d'une fonction sémantique est la suivante :

> **Définition 3 (fonction sémantique)** : une fonction sémantique représente la connexion entre la syntaxe et la sémantique. Elle permet de faire correspondre un objet du monde syntaxique vers un objet du monde sémantique.

Une fonction sémantique est définie par sa signature et par des équations sémantiques. La *signature* d'une fonction est notée $f : A \rightarrow B$, ce qui signifie que f est un membre de $A \rightarrow B$, en sachant que $A \rightarrow B$ dénote l'ensemble des fonctions avec A comme domaine et B comme co-domaine. Par exemple, la fonction $eval : Expr \rightarrow Number$ prend une expression et produit un entier.

La sémantique des fonctions est exprimée par une ou plusieurs *équations sémantiques*. Par exemple, la sémantique du langage d'expressions ci-dessus, pour la fonction $eval$, est donnée par les équations suivantes :

$$eval : Expr \rightarrow Number$$
$$eval\ [\![e_1 + e_2]\!] = eval\ [\![e_1]\!] + eval\ [\![e_2]\!]$$
$$eval\ [\![e_1 - e_2]\!] = eval\ [\![e_1]\!] - eval\ [\![e_2]\!]$$
$$eval\ [\![i]\!] = i$$

La notation $eval\ [\![e]\!]$ dénote le sens de l'expression e. Entre ces crochets sont placés les objets syntaxiques d'un langage. Ces crochets permettent de différencier la syntaxe de la sémantique (l'addition et la soustraction à droite). À titre d'exemple, la première équation signifie que la valeur de l'expression syntaxique $e_1 + e_2$ est calculée en évaluant l'expression e_1 et l'expression e_2 et en opérant l'addition de ces deux valeurs. Par la suite, pour plus de lisibilité, les crochets sémantiques $[\![$ et $]\!]$ sont remplacés par des crochets simples $[$ et $]$.

La fonction $eval$ précédente n'a pas besoin d'un contexte pour être évaluée. Considérons la syntaxe suivante des expressions augmentée avec le nom de variables :

$$i : Number$$
$$n : String$$
$$e : Expr \qquad e ::= e + e \mid e - e \mid i \mid n$$

L'évaluation de cette syntaxe dépend d'un environnement ou *contexte* ρ constitué d'un ensemble de variables et des valeurs associées. Ce contexte est défini par la fonction suivante :

$$\rho : Context = String \rightarrow Number$$

La sémantique dénotationnelle de cette syntaxe augmentée prend alors une expression, un contexte et retourne un nombre. Elle est donc définie par les équations suivantes :

$$eval : Expr \rightarrow Context \rightarrow Number$$
$$eval[e_1 + e_2]\rho = eval[e_1]\rho + eval[e_2]\rho$$
$$eval[e_1 - e_2]\rho = eval[e_1]\rho - eval[e_2]\rho$$
$$eval[i]\rho = i$$
$$eval[n]\rho = \rho(n)$$

La notation $eval[e]\rho$ signifie que l'expression e est évaluée dans le contexte ρ. Par exemple, la dernière équation signifie que l'évaluation de la variable n dans le contexte ρ est donnée par l'évaluation de la fonction ρ appliquée à la variable n.

Nous verrons une première utilisation de la sémantique dénotationnelle dans la section 2.3.

2.2. Fonctionnement interne de XSLT

Le langage XSLT comporte quelques traits de fonctionnement différents des autres langages notamment parce qu'il mélange plusieurs paradigmes de programmation : c'est un langage à base de règles et fonctionnel qui offre des fonctions pures [Kay 00]. XSLT permet de déclarer des variables avec une valeur initiale mais ne permet pas de modifier cette valeur. Nous détaillons les possibilités et le fonctionnement de XSLT dans les sous-sections suivantes, en commençant par rappeler le vocabulaire et le processus de base à travers un exemple.

2.2.1. XSLT par l'exemple

Nous rappelons le vocabulaire associé au contenu d'une feuille de transformation et décrivons le fonctionnement d'un processeur de transformation à travers un exemple. Il nous servira de référence tout au long de ce chapitre. Le document source de notre exemple est le suivant :

```
1. <?xml version="1.0" encoding="ISO-8859-1"?>
2. <article>
3.     <title>Incremental transformation </title>
```

```
4.    <artheader>
5.       <authorgroup>
6.          <author>
7.             <firstname>Lionel</firstname>
8.             <lastname>Villard</lastname>
9.          </author>
10.            <author>
11.               <firstname>Nabil</firstname>
12.               <lastname>Layaïda</lastname>
13.            </author>
14.         </authorgroup>
15.         <date>28 October 2000</date>
16.      </artheader>
17.      <section>
18.         <title>Introduction</title>
19.         <para>…</para>
20.      </section>
21.      <section>
22.         <title>Toward a WYSIWYG edition of XML Documents</title>
23.         <para>…</para>
24.         <section>
25.            <title>An example</title>
26.            <para>…</para>
27.         </section>
28.         <section>
29.            <title>Process overview</title>
30.         </section>
31.         </section>
32.      </article>
```

Ce document XML est une instance de la classe de document Docbook (cf. section 2.2.2). Une présentation possible sur un écran est illustrée dans la figure 61. Cette figure est suivie par le code HTML correspondant à cette présentation. La figure 61 montre la table des matières de l'article précédée du nom du premier auteur et du titre de l'article. En bas de l'écran, la liste complète des auteurs est présentée, suivi du nombre de sections de plus haut niveau et la date de la dernière modification du document.

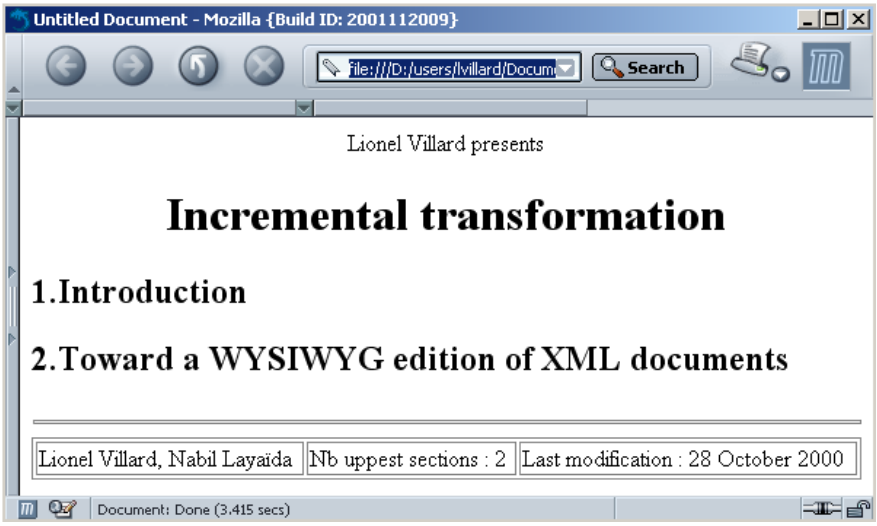


Figure 61. Une présentation possible d'une table des matières d'un article.

```
1. <html>
2.     <body>
3.         <p align="center">Lionel Villard  presents</p>
4.         <h1 align="center">Incremental transformations with XSLT</h1>
5.         <h2 align="left" style="padding-left=0px">1. Introduction</h2>
6.         <h2 align="left" style="padding-left=0px">2. Toward a WYSIWYG edition of XML
               Documents </h2>
7.         <hr>
8.         <table width="100%" border="1">
9.             <tr>
10.                <td>Lionel Villard and Nabil Laya&iuml;da</td>
11.                    <td>Nb uppest sections : 2</td>
12.                    <td>Last modification : 28 October 2000</td>
13.                </tr>
14.            </table>
15.        </body>
16.    </html>
```

La présentation ci-dessus est le résultat de la transformation appliquée sur la feuille de transformation donnée en annexe A. Nous donnerons des extraits de cette feuille tout au long de ce chapitre, en respectant la numérotation des lignes de la feuille complète de l'annexe.

Lorsque l'exécution de la transformation débute, le processeur de transformation recherche une *règle*, représentée par l'élément template, pour la racine du document source. La racine du document est dans notre cas l'élément article (la vraie racine d'un document est le nœud de type document que nous ne considérons pas ici). Pour chacune des règles, le processeur teste si le pattern associé à la règle correspond à l'élément article. Un *pattern* est un patron caractérisant un ensemble de nœuds. Par exemple le pattern article caractérise l'ensemble des éléments de type article. La règle trouvée est ensuite *instanciée*, c'est-à-dire exécutée avec comme *nœud source courant* l'élément article. Dans notre exemple, la règle suivante est instanciée :

```
8.  <xsl:template match="article">
9.      <html>
10.         <body>
…
17.             <h1 align="center"><xsl:value-of select="title"/></h1>
18.             <xsl:apply-templates select="section"> … </xsl:apply-templates>
…
22.             <table width="100%" border="1">
…
28.             </table>
29.         </body>
30.     </html>
31.  </xsl:template>
```

Le corps de chaque règle est composé d'une suite d'*instructions*. L'exécution de ces instructions est effectuée séquentiellement. Chaque instruction produit une action particulière. Dans notre exemple, les éléments html et body sont copiés dans le document cible, suivi par la génération de l'en-tête. Le titre est présenté dans un élément h1. Puis la table des matières est générée grâce à l'instruction apply-templates (cf. ci-dessous). La table html qui contient la liste

complète du nom des auteurs, le nombre de sections de plus haut niveau et la date de la dernière modification du document est ensuite générée.

L'exécution de l'instruction apply-templates ligne 18 consiste dans un premier temps à sélectionner un ensemble de nœuds dans le document source. Cette sélection est réalisée grâce à une *expression* spécifiée dans le langage XPath. Ensuite pour chacun de ces nœuds, les règles qui correspondent au nœud considéré sont recherchées dans la feuille de transformation. Seule la plus spécifique (repose sur la notion de priorité, cf. la spécification XPath) est conservée. Cette instruction est une des plus importantes du langage XSLT, c'est pourquoi nous étudions son action, appelé *processus d'instanciation*, plus en détail dans la section 2.2.3. Dans notre exemple, l'instruction apply-templates sélectionne un ensemble de sections à partir du nœud source courant article, c'est-à-dire les éléments des lignes 17 et 21 du document source. La règle qui correspond le mieux à ce type de nœud est la suivante :

```
33.     <xsl:template match="section">
34.         <xsl:param name="indent">0</xsl:param>
35.
36.         <xsl:variable name="heading">
37.             <xsl:choose>
…                   <!-- Depending on the section depth, choose a heading tag -->
42.             </xsl:choose>
43.         </xsl:variable>
44.
45.         <xsl:element name="{$heading}">
46.             <xsl:attribute name="style">padding-left=
47.                 <xsl:value-of select="$indent"/>px
48.             </xsl:attribute>
…       …
51.             <xsl:number value="position()" format="1."/>
52.             <xsl:text> </xsl:text>
53.             <xsl:value-of select="title"/>
54.         </xsl:element>
55.
56.         <xsl:if test="count(ancestor::section) &lt; $toc.depth - 1">
57.             <xsl:apply-templates select="section">
58.                 <xsl:with-param name="indent" select="$indent + 100"/>
…                   …
61.         </xsl:if>
62.     </xsl:template>
```

Cette règle est déclarée avec un paramètre nommé indent qui par défaut est initialisé à zéro. L'exécution de cette règle commence par la création de la variable heading (ligne 36). Sa valeur contient le nom de l'élément html associé à la profondeur de la section (h2 pour le niveau 0, h3 pour le niveau 1, etc.). Ensuite un élément dont le nom est contenu dans la variable heading est généré dans l'arbre cible (ligne 45). L'attribut style est ajouté à cet élément avec comme valeur un décalage à gauche contenu dans le paramètre indent. La numérotation de la section est ensuite générée (ligne 51), suivie par le titre de la section (ligne 53). Puis en fonction de la profondeur de la section en cours d'instanciation, les sections descendantes sont instanciées (ou non).

La spécification de la transformation peut être répartie en plusieurs *modules*. Par la suite, une feuille de transformation est considérée comme étant la spécification entière de la transformation, c'est-à-dire la réunion de l'ensemble des modules.

Les points importants à retenir du langage XSLT sont les suivants :

- l'exécution d'une instruction s'effectue pour un nœud source courant. Ce nœud est un des paramètres qui compose le *contexte d'exécution* d'une instruction. D'autres paramètres constituent ce contexte comme nous le verrons dans la section 2.2.2 ;
- le processus le plus complexe de XSLT est certainement le processus d'instanciation. En effet, c'est ce processus qui remet en cause de façon importante le contexte d'exécution (cf. section 2.2.3) ;
- chaque instruction a un comportement particulier lors de son exécution. Pour éviter de considérer chacune de ces instructions lors de la conception de notre processeur incrémental, il est nécessaire de les classer. Nous proposons donc une taxonomie dans la section 2.2.4 ;
- l'historique produit par l'exécution d'une transformation peut être stocké dans une structure appelée arbre d'exécution (cf. chapitre 5 section 2.2.2). Un fragment de l'arbre d'exécution correspondant à notre exemple est donné dans la section 2.2.5.

2.2.2. Modélisation du contexte d'exécution

L'exécution de chaque instruction d'un programme XSLT Π s'effectue dans un *contexte d'exécution*. Ce contexte est composé d'un contexte statique et d'un contexte dynamique. Le contexte statique est composé :

- de l'ensemble des déclarations des espaces de noms visibles au moment de l'évaluation de l'instruction. Par exemple, la déclaration de l'attribut `xmlns:db="http://www.oasis-open.org/docbook/..."` est un constituant du contexte statique ;
- de l'ensemble des déclarations de variables ou de paramètres visibles au moment de l'évaluation de l'instruction.

Le contexte dynamique est un tuple $P = (n_s \in t_\Sigma, l_s, p_s, V, r_t \in R, i_t \in R, n_c \in t_\Delta, p_c)$ correspondant aux définitions suivantes :

- la position courante dans le document source. Cette position est composée :
  - du nœud source courant n_s : c'est le nœud source qui est en train d'être instancié ;
  - de la liste courante de nœuds source l_s : lorsqu'une instruction `apply-templates` ou `for-each` est exécutée, une liste de nœuds est sélectionnée dans le document source et devient la liste de nœuds courante. Cette liste est utilisée lors de l'évaluation de la fonction `last()`, qui retourne la position du dernier nœud de la liste ;
  - de la position courante p_s qui indique la position du nœud source courant dans la liste courante de nœuds source ;

- l'ensemble V correspondant à la valeur courante des variables visibles par l'instruction courante ;
- la position courante dans la feuille de transformation. Cette position est composée :
    - de la règle courante r_t avec son mode associé et éventuellement son nom et son pattern ;
    - de l'instruction courante i_t, c'est-à-dire celle qui est en cours d'exécution ;
- la position courante dans le document cible. Cette position est composée :
    - du nœud cible courant n_c : c'est le nœud contextuel sous lequel de nouveaux nœuds sont ajoutés lors de l'exécution d'une instruction ;
    - de la position courante p_c correspondant à la position du prochain nœud fils inséré sous le nœud cible. Lorsque le nœud courant cible est un texte, alors la position p_c correspond au nombre de caractères qui ont été généré.

Chaque exécution d'une instruction s'effectue dans un contexte d'exécution $\rho \in P_\Pi$, avec P_Π correspondant à l'ensemble des contextes utilisés lors de l'exécution du programme XSLT Π. Pour une même instruction, plusieurs contextes d'exécution lui sont associés. Par exemple, pour un document source contenant une suite de sections, chaque application d'une règle sélectionnant une section produit un contexte d'exécution différent puisque, en particulier, le nœud source courant n_s est différent. Par contre, le contexte statique reste le même puisqu'il dépend uniquement de la position de l'instruction dans la feuille de transformation. C'est pourquoi la plupart des processeurs de transformation actuels effectuent une évaluation partielle (cf. chapitre 2 section 4.3) de chaque expression utilisant le contexte statique pour augmenter les performances.

2.2.3. Processus d'instanciation

Le processus d'instanciation d'une règle est illustré figure 62. La première étape consiste en la sélection d'un ensemble de nœuds source qui devient alors la liste courante l_s des nœuds source sélectionnés. Ensuite, pour chacun de ces nœuds, une règle est recherchée parmi l'ensemble des règles déclarées dans la feuille de transformation. Un des critères de recherche est la correspondance du nœud source en train d'être traité et le pattern de la règle. Si une règle est trouvée pour ce nœud, alors le nœud source courant n_s devient le nœud en train d'être traité et la position courante p_s est celle du nœud traité dans la liste l_s. Ensuite la règle est exécutée. À la fin de l'exécution de la règle, le nœud suivant dans la liste, s'il existe, est considéré.

Pour résumer, le processus d'instanciation entraîne les modifications suivantes sur le contexte d'exécution :

- il établit une nouvelle liste courante l_s ;
- il change le nœud source courant n_s et par conséquent la position courante p_s du nœud traité ;
- la règle courante i_t est modifiée.

La modification du document source peut entraîner le changement de chacun de ces paramètres. Les deux premiers paramètres peuvent changer car ils sont issus du résultat de l'évaluation d'une expression. Le dernier paramètre peut changer car la recherche d'une règle dépend du document source. En effet, considérons par exemple l'extrait d'une transformation XSLT composée des deux règles suivantes :

```
1. <xsl:template match="section"> … </xsl:template>
2. <xsl:template match="section[title]"> … </xsl:template>
```

La première règle est instanciée pour un élément **section** qui ne contient pas d'élément de type **title**. Par contre, si un élément type **title** est ajouté ou supprimé de ce nœud, alors c'est la seconde règle qui est instanciée car elle est plus spécifique. Par conséquent, une instruction **apply-templates** qui sélectionne des nœuds de type **section** nécessite potentiellement d'être réexécutée.

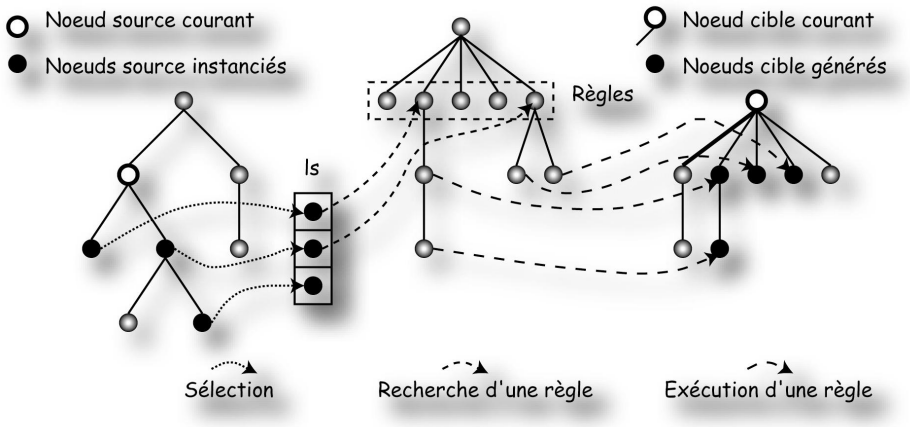


Figure 62 Processus d'instanciation.

Il faut noter que le processus d'instanciation modifie complètement le contexte dynamique lié à la position courante dans le document source. Il modifie également l'ensemble des variables et la position courante dans la feuille de transformation. La portée d'un contexte donné est donc locale à une règle de transformation.

2.2.4. Taxonomie des instructions

Dans ce chapitre nous voulons proposer une solution couvrant le plus possible l'ensemble de la spécification XSLT. Pour éviter d'avoir à traiter toutes les instructions une par une, nous proposons une taxonomie de ces instructions. L'intérêt de cette taxonomie est double. Premièrement, elle permet de regrouper dans une seule classe les instructions sur lesquelles les mêmes traitements sont appliqués. De plus, la prise en compte de nouvelles instructions, avec notamment l'arrivée de XSLT 2.0 [XSLT2.0 01], sera simplifiée par cette taxonomie.

La taxonomie que nous proposons repose sur la dépendance de l'exécution d'une instruction vis-à-vis du contexte dynamique. Le tableau 3 donne la liste exhaustive des instructions du langage XSLT 1.0 qui sont dépendantes du contexte dynamique. Nous donnons leur relation vis-à-vis de chaque paramètre de ce contexte. Cette relation se caractérise par une relation en lecture seulement (L), en écriture seulement (E) ou en lecture et écriture (L/E). Si aucune relation n'existe entre un paramètre du contexte et une instruction la case est laissée vide. De

plus nous indiquons dans la colonne Expr lorsqu'une instruction dépend d'une expression (Expr) et/ou d'un pattern (Pat).

À partir de ce tableau nous avons identifié quatre catégories d'instruction. Ces catégories sont différenciées dans le tableau par une alternance de couleurs, gris clair et gris foncé, indiquant le changement de catégorie. Ces catégories sont les suivantes :

- l'instruction template, c'est la seule instruction non opérationnelle. Elle ne fait que déclarer une règle. Son utilisation est effectuée par les autres instructions ;
- les instructions de type *contrôle* qui modifient la position de l'instruction courante i_t autrement que par un cheminement purement séquentiel. Il faut noter que l'exécution de certaines de ces instructions dépend du résultat de l'expression associée. Par exemple le contenu de l'instruction if est exécuté si l'expression associée est évaluée à vraie ;
- les instructions de type *production* qui modifient le contexte cible. Par exemple, l'instruction value-of génère des caractères dans l'arbre cible, la position courante correspondant au nombre de caractères est alors mise à jour ;
- les instructions de type *variable* (les instructions variable et param) sont les seules à modifier les variables du contexte.

Grâce à cette taxonomie nous pouvons restreindre notre étude à un sous-ensemble d'instructions XSLT.

Instructions XSLT	**Contexte dynamique**								
	Position source			**Position transformat.**		**Position cible**		**Variables**	**Expr/ Pat**
	n_s	l_s	p_s	r_t	i_t	n_c	p_c		
template									Pat
apply-imports	L			L	E				
apply-templates	L/E	E	E		E			L	Expr
call-template					E				
choose					E				
for-each	L/E	E			E			L	Expr
if	L	L	L		E			L	Expr
sort		L/E			(E)			L	Expr
otherwise					(E)				
when	L	L	L		(E)			L	Expr
with-param	L	L	L		(E)			L	Expr
attribute						L/E		L	Expr
comment						L/E	L/E	L	
copy	L					L/E	L/E	L	
copy-of	L					L/E	L/E	L	Expr
element						L/E	L/E	L	Expr

processing-instruction						L/E	L/E	L	Expr
number	L	L	L			L/E	L/E	L	Expr/ Pat
text						L/E	L/E		
value-of	L	L	L			L/E	L/E	L	Expr
param	L	L	L					L/E	Expr
variable	L	L	L					L/E	Expr

Tableau 3. Taxonomie des instructions XSLT.

2.2.5. Arbre du flot d'exécution

L'historique de l'exécution d'une transformation est stocké dans un arbre représentant le flot de l'exécution. En ce qui concerne XSLT, l'historique est constitué principalement par les contextes d'évaluation de chaque instruction exécutée, par le résultat de l'évaluation de chaque expression évaluée et par l'instanciation des règles.

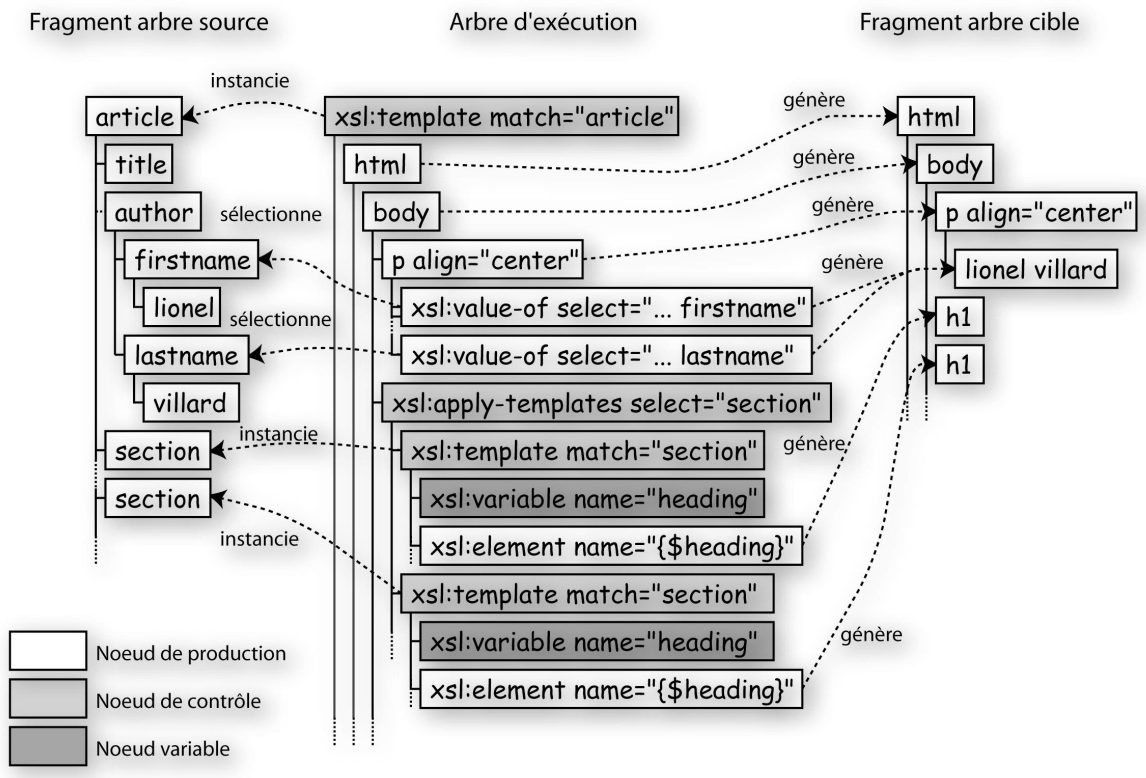


Figure 63. Arbre représentant l'exécution d'un programme XSLT.

La figure 63 représente au centre un fragment de l'arbre d'exécution de l'exemple de ce chapitre. Chaque nœud, appelé *nœud d'exécution*, représente une exécution particulière d'une instruction. Sur chacun de ces nœuds, une ou plusieurs informations peuvent être mémorisées pour stocker les paramètres de l'historique. Par exemple, les nœuds d'exécution de type `template` peuvent conserver le nœud source instancié. Les instanciations sont conservées grâce aux liens hiérarchiques entre les nœuds d'exécution de type `apply-templates` et

templates. Grâce à ces liens et aux nœuds source stockés sur les nœuds d'exécution de type template, il est possible de restaurer la liste courante des nœuds l_s.

La taille mémoire requise pour stocker cet arbre dépend des informations conservées et de celles qui sont au contraire recalculées. Nous verrons quelques évaluations de cette taille dans la section 4.

Nous avons vu dans le chapitre 2 section 4.3 lors de l'étude des algorithmes incrémentaux que l'une des techniques pour rendre un algorithme incrémental était de réutiliser les calculs intermédiaires. Ces calculs sont dans notre cas stockés dans cet arbre.

2.3. Sémantique de XPath

Dans le langage XSLT, de nombreuses instructions utilisent une expression (cf. tableau 3 de la section précédente) soit pour sélectionner un ensemble de nœuds source ou soit pour effectuer des calculs divers au sein d'une instruction. Donc le cœur de tout algorithme d'évaluation incrémentale pour XSLT passe par l'étude fine du langage d'expression XPath

Dans cette section, nous décrivons la sémantique de ce langage en utilisant la notation introduite dans [Wadler 00] qui est basée sur la sémantique dénotationnelle (cf. section 2.1.2). Pour cela, nous commençons par définir la syntaxe abstraite du langage. Puis la sémantique relative à l'évaluation d'une expression est décrite. Nous étudions ensuite la sémantique des patterns utilisés en particulier dans les instructions template à travers l'attribut match, dont la syntaxe est identique à celle d'une expression mais avec les restrictions suivantes :

- le résultat de l'évaluation d'une expression doit être un ensemble de nœuds ;
- seuls les axes child et attribute sont permis.

2.3.1. Syntaxe abstraite d'une expression XPath

La syntaxe abstraire d'une expression est la suivante :

q : Name
s : String
i : Integer
nt : NodeTest
*nt ::= q | * | text() | node()*
a : Axis
a ::= child | parent | ancestor | descendant
st :Step
st ::= a :: nt | a :: nt [e]
p : Path
p ::= lp | fe
lp : LocPath
lp ::= st | lp_1 / lp_2
fe : FilterExpr
fe ::= fn((e (, e))?) | fe [e] | s | i*
fn : FuncName
fn ::= position | last | not | name | …
op : Operator

$$op ::= \textbf{or} \mid \textbf{and} \mid + \mid - \mid < \mid > \mid * \mid + \mid \ldots$$

$$e : Expr$$

$$e ::= e_1\ op\ e_2 \mid p \mid p_1 \mid p_2$$

Une expression e est soit une composition d'expressions reliées par des opérateurs op, soit un chemin p ou une union de chemins. Deux types de chemins peuvent être spécifiés. Le premier est un *chemin de localisation* (LocPath), ou aussi appelé *sélecteur*, composé d'une succession d'*étapes* (Step). Chaque étape est constituée d'un *axe* a, d'un *test de nœud* nt et éventuellement d'une expression e appelée *prédicat*. Le test de nœud q dénote un type d'élément particulier, tandis que le caractère * dénote n'importe quel nœud de type élément. Les constructeurs $text()$ et $node()$ indiquent respectivement un nœud de type texte et n'importe quel nœud. Le deuxième type de chemin est un filtre fe qui peut être soit l'appel à une fonction, soit un filtre avec un prédicat, soit un littéral ou soit un nombre.

Nous avons volontairement omis certains constructeurs syntaxiques du langage afin de rendre la suite de ce chapitre plus claire. En particulier, les espaces de noms ne sont pas décrits par cette syntaxe. De plus, nous restreignons notre étude à un sous-ensemble de tests sur les nœuds et un sous-ensemble des axes. XPath permet de spécifier plusieurs prédicats, alors que notre syntaxe n'en permet qu'un seul. Les variables ne sont pas prises en compte. Enfin, seuls les chemins de localisation relatifs sont pris en compte.

Nous définissons quelques domaines et fonctions utiles dans la suite de ce chapitre. Pour commencer, nous définissons le domaine $Node$. Ce domaine représente l'ensemble des nœuds d'un arbre XML. La notation $n : Node$ signifie que n est un membre de $Node$, c'est-à-dire que c'est un nœud. Nous définissons les fonctions suivantes sur le domaine $Node$:

$$name : Node \rightarrow String$$
$$type : Node \rightarrow String$$
$$isElement() : Node \rightarrow Boolean$$
$$isText() : Node \rightarrow Boolean$$

La première fonction prend un nom et retourne le nom de ce nœud, tandis que la deuxième fonction retourne le type de ce nœud (texte, élément, attribut, etc.). La troisième fonction teste si un nœud est un élément et la dernière teste si un nœud est un texte.

Nous définissons les fonctions suivantes relatives aux axes de parcours :

$$child : Node \rightarrow Set(Node)$$
$$parent : Node \rightarrow Set(Node)$$
$$ancestor : Node \rightarrow Set(Node)$$
$$descendant : Node \rightarrow Set(Node)$$

La fonction $child$ retourne la liste des nœuds fils du nœud passé en argument. Les autres fonctions se comprennent facilement. Grâce à ces fonctions, nous pouvons définir la fonction dénotant la sémantique des axes :

$$axis : Axis \rightarrow Node \rightarrow Set(Node)$$
$$axis[child]n = \{ m \in t_\Sigma / m \in child(n) \}$$
$$axis[parent]n = \{ m \in t_\Sigma / m \in parent(n) \}$$
$$axis[ancestor]n = \{ m \in t_\Sigma / m \in ancestor(n) \}$$
$$axis[descendant]n = \{ m \in t_\Sigma / m \in descendant(n) \}$$

Cette fonction, à partir d'un nom d'axe et d'un nœud n récupère une liste de nœuds. Une propriété intéressante sur les axes est la suivante :

> **Proposition 1 (axe inverse)** : à un axe donné correspond toujours un axe inverse.

La preuve de cette proposition consiste à énumérer l'ensemble des axes avec l'axe inverse correspondant :

$$n_1 \in child(n_2) \Leftrightarrow n_2 \in parent(n_1)$$
$$n_1 \in ancestor(n_2) \Leftrightarrow n_2 \in descendant(n_1)$$

Cette proposition est vraie sur l'ensemble des axes du langage XPath à condition de l'étendre par un axe inverse pour les axes attribute et namespace (que nous ne considérons pas). Ces deux axes sélectionnent des attributs, l'axe inverse consiste à sélectionner leur parent.

Grâce à cette proposition, nous pouvons définir la fonction $axis^{-1}$ qui correspond à la sélection inverse de l'axe passé en argument.

2.3.2. Sémantique des expressions

L'évaluation d'une expression XPath dépend du contexte statique et en partie du contexte dynamique. Les paramètres nécessaires du contexte dynamique sont ceux modélisant le contexte du document source, c'est-à-dire le nœud source courant n_s, la liste courante de nœuds l_s et la position du nœud courant p_s dans cette liste. Comme notre syntaxe ne permet pas de spécifier des variables, le paramètre V n'est pas nécessaire. De façon formelle, nous définissons un contexte Ctx comme suit :

$$Ctx = Node \times Set(Node) \times Integer$$

avec $Node$ le domaine du nœud source, $Set(Node)$ le domaine de la liste courante et $Integer$ le domaine de la position du nœud courant. Par la suite, nous notons un environnement $\rho : Ctx$ tel que $\rho = (n, l, p)$. Comme il n'y a pas d'ambiguïté avec le document cible, l'indice source s est supprimé. En fait, nous le remplaçons par un nouvel indice représentant la projection d'un contexte sur un des paramètres du contexte. Par exemple n_ρ représente la projection du contexte ρ sur le nœud courant.

La première fonction sémantique que nous définissons et la fonction $nodeTest$ qui renvoie vrai si un nœud correspond au test :

$nodeTest : NodeTest \rightarrow Node \rightarrow Boolean$
$nodeTest[q]n$ = vrai si name(n) = q, faux sinon
$nodeTest[*]n$ = isElement(n)
$nodeTest[text()]n$ = isText(n)
$nodeTest[node()]n$ = vrai

Sur ces expressions, deux fonctions sémantiques peuvent être définies :

$$eval\text{-}ns : Expr \rightarrow Ctx \rightarrow Set(Node)$$
$$eval\text{-}bo : Expr \rightarrow Ctx \rightarrow Boolean$$

La fonction ***eval-ns*** (ns : nodeset) correspond à l'évaluation d'une expression dont le résultat est un ensemble de nœuds. La seconde fonction ***eval-bo*** évalue une expression dont le résultat est un booléen.

Les constructions syntaxiques pouvant être dénotées par la première fonction sont les sélecteurs et l'union de sélecteurs. Par définition, les équations sémantiques de la première fonction sont les suivantes :

$$
\begin{aligned}
&\textit{eval-ns[a :: nt]}\rho = \{\, n_1 \in t_\Sigma \,/\, n_1 \in \textit{axis[a]}n_\rho \wedge \textit{nodeTest[nt]}n_1 \,\} \\
&\textit{eval-ns[a :: nt[e]]}\rho = \{\, n_1 \in t_\Sigma \,/\, n_1 \in \textit{axis[a]}n_\rho \wedge \textit{nodeTest[nt]}n_1 \\
&\qquad \wedge\ \textit{eval-bo[e]}\rho_2 \,\} \\
&\qquad \textit{avec } n_{\rho 2} = n_1 \textit{ et } l_{\rho 2} = \textit{eval-ns[a :: nt]}\rho. \\
&\textit{eval-ns[}lp_1 \,/\, lp_2\textit{]}\rho = \{\, n_2 \in t_\Sigma \,/\, n_1 \in \text{eval-ns}\textit{[}lp_1\textit{]}\rho, \\
&\qquad n_2 \in \textit{eval-ns[}p_2\textit{]}\rho_1 \,\} \\
&= \textit{(eval-ns[}lp_1\textit{]} \,;\, \textit{eval-ns[}lp_2\textit{])}\rho \\
&\qquad \textit{avec } n_{\rho 1} = \textit{eval-ns[}lp_1\textit{]}\rho \textit{ et } n_{\rho 1} \in l
\end{aligned}
$$

Pour évaluer un prédicat, il est nécessaire de définir les équations sémantiques de la seconde fonction. Ces équations sont les suivantes :

$$
\begin{aligned}
&\textit{eval-bo[a :: nt]}\rho = \text{vrai si } \exists\, n \in \textit{axis[a]}n_\rho \wedge \textit{nodeTest[nt]}n \\
&\qquad\qquad\qquad\qquad\ \textit{faux sinon} \\
&\textit{eval-bo[position() = i]}\rho = p_\rho = i \\
&\textit{eval-bo[position() = s]}\rho = p_\rho = \text{number(s)}
\end{aligned}
$$

avec la fonction ***number*** qui permet de convertir une chaîne de caractères vers un entier selon les règles définies par le langage XPath.

2.3.3. Sémantique des patterns

Le pattern défini par la fonction match est utilisé pour caractériser un ensemble de nœuds. Cette fonction est associée aux équations sémantiques suivantes :

$$
\begin{aligned}
&\textit{match} : \textit{Expr} \rightarrow \textit{Node} \rightarrow \textit{Boolean} \\
&\textit{match[nt]}n = \textit{nodeTest[nt]}n \\
&\textit{match[child :: nt]}n = \text{vrai si } \textit{nodeTest[nt]}n \\
&\textit{match[child :: nt[e]]}n = \text{vrai si } \textit{nodeTest[nt]}n \wedge \textit{eval-bo[e]}\rho \\
&\qquad\qquad\qquad \textit{avec } n_\rho = n \textit{ et } l_\rho = \textit{eval-ns[a :: nt]}n
\end{aligned}
$$

Comme la syntaxe d'un pattern est identique à celle des expressions, la fonction ***match*** prend une expression, un nœud est retourne un booléen. Lorsque le pattern est évalué à vrai alors le nœud testé correspond au pattern.

2.3.4. Conversion de type

Le résultat d'une expression peut être converti dans un autre type. Le type de conversion dépend notamment du contexte d'utilisation de l'expression. Par exemple, la figure 64 montre la conversion d'un ensemble de nœuds vers une chaîne de caractères. Dans cet exemple, on peut noter que le résultat final ne dépend pas uniquement du résultat de l'évaluation de l'expression `date` : il dépend aussi du résultat de la conversion qui, dans ce cas, sélectionne l'ensemble des nœuds texte descendant du premier nœud sélectionné par l'expression `date`.

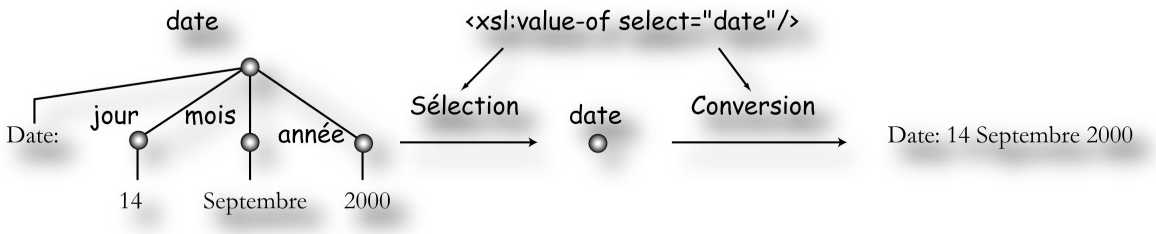


Figure 64. Evaluation de l'expression `date` et conversion vers une chaîne de caractères.

Le langage XPath définit de nombreuses règles de conversion entre les différents types primitifs du langage (nombre, chaîne de caractères, booléen, ensemble de nœuds et fragments d'arbres). Cependant, seule la conversion d'un ensemble de nœuds vers une chaîne de caractères nécessite de considérer des nœuds du document source autres que les nœuds sélectionnés par l'expression elle-même. Nous introduisons formellement ce type de conversion par la fonction suivante :

$$\begin{aligned} &nodeToString && : Set(Node) \rightarrow String \\ &l && : Set(Node) \\ &nodeToString(l) && = \{\, x \,/\, x \in descendant(l[1]) \wedge type(x) = text() \,\} \end{aligned}$$

2.4. Synthèse

Dans cette section nous avons décrit le fonctionnement d'un processeur de transformation XSLT. En particulier, nous avons défini de façon précise le contexte d'exécution d'une instruction et nous avons identifié les instructions agissant sur ce contexte. De plus, nous avons décrit de façon formelle le langage d'expression XPath en utilisant la sémantique dénotationnelle. Nous sommes en mesure maintenant de décrire le fonctionnement de notre processeur de transformation incrémental.

3. Processeur incrémental

L'objet de cette section est de retracer chacune des étapes permettant de concevoir un processeur de transformation incrémental. Pour cela, un exemple est d'abord exposé afin de donner un aperçu de comment la mise à jour incrémentale peut être effectuée. Ensuite, nous identifions les conséquences de l'édition sur l'exécution de la transformation. Ces conséquences vont nous amener à considérer les quatre fonctions que doit implanter un processeur incrémental : la détection statique des expressions à réévaluer, la remise en cause des instanciations, le contrôle des répercutions des réévaluations et l'exécution de la transformation de façon incrémentale.

3.1. Exemple

Supposons que l'auteur modifie le document source en insérant une nouvelle section dans le document décrit dans la section 2.2.1. Dans ce cas, les conséquences sur le document cible sont les suivantes : un élément `h2` doit être généré avec le numéro correspondant. De plus, en fonction de l'endroit où est insérée la section, les sections suivantes doivent être renumérotées. Le compteur de section de plus haut niveau doit aussi être mis à jour. Au niveau de la feuille de transformation, les instructions qui nécessitent d'être réexécutées sont les suivantes :

- l'instruction apply-templates qui sélectionne les éléments section (ligne 18) : la règle ligne 33 doit être appliquée dans sa globalité avec le nœud section nouvellement inséré comme nœud source courant ;
- l'instruction number (ligne 51) pour l'ensemble des sections suivant la nouvelle section ;
- la deuxième cellule de la table html (ligne 25).

Ces instructions constituent la liste optimale car minimale pour mettre à jour le document cible après l'ajout d'une section. Nous verrons dans cette section comment cette liste est calculée.

3.2. Modification des règles de transformation en phase d'édition

Dans cette section, nous identifions les conséquences d'abord de la modification du document source puis de la spécification de la transformation sur l'exécution de la transformation.

3.2.1. Édition du document source

La modification du document source entraîne comme conséquence directe la modification du résultat de l'évaluation de certaines des expressions contenues dans la feuille de transformation (cf. figure 65). Pour être précis, seules les expressions nécessitant la connaissance du contexte source (le nœud source, la liste des nœuds source et la position courante) ont potentiellement besoin d'être réévaluées. Par exemple, l'expression round(43/2) + string-length("slide") ne dépend pas du contexte source et donc son évaluation ne sera jamais affectée par la modification du document source. La première difficulté est donc de détecter quelles sont les expressions qui, au contraire, nécessitent d'être réévaluées.

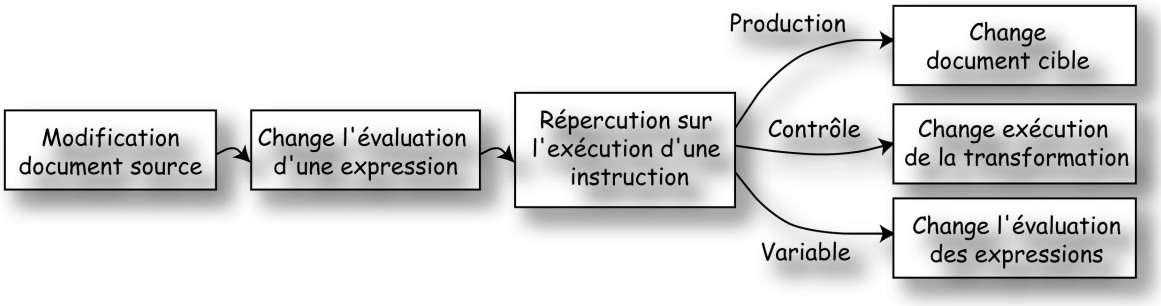


Figure 65. Conséquences d'une modification sur le document source.

En plus des conséquences directes, c'est-à-dire le changement de l'évaluation d'une expression, cette réévaluation entraîne des répercussions sur l'exécution de l'instruction associée à cette expression. Le type de répercussion dépend du type de l'instruction associée à l'expression (selon la taxonomie établie dans la section 2.2.4) :

- type contrôle : ces instructions modifient le contexte d'exécution en fonction du résultat de l'expression associée. Les répercussions sont de natures différentes selon le paramètre du contexte dynamique modifié :
    - liste courante de nœuds l_s : le changement de ce paramètre entraîne des répercussions sur les expressions qui utilisent ce paramètre ainsi que la position p_s des nœuds dans cette liste. Cette répercussion est limitée à l'ensemble des

nœuds d'exécution qui ont une relation de descendance avec l'instruction réévaluée ;

- instruction courante i_t : certaines instructions de contrôle orientent le processus de transformation en fonction de l'expression associée, comme l'instruction if. Par conséquent, une partie de la transformation peut nécessiter d'être exécutée alors qu'elle ne l'était pas, ou au contraire ne plus être exécutée. Pour l'instruction apply-templates, les instanciations qu'elle génère (cf. section 2.2.3) peuvent potentiellement être remises en cause ;

- type production : le document cible doit être mis à jour ;

- type variable : la modification de la valeur d'une variable entraîne la modification du contexte d'exécution. Par conséquent, toutes les expressions contenant une référence vers la variable concernée doivent être réévaluées.

3.2.2. Édition de la transformation

Lorsque la feuille de transformation est modifiée, les conséquences directes sur l'exécution de celle-ci sont les suivantes, selon le type d'opération effectué :

- changement de la valeur d'une variable : les conséquences sont les mêmes que précédemment pour le cas type de variable ;

- ajout ou suppression d'une instruction template : les instanciations effectuées par l'instruction apply-templates peuvent potentiellement être remises en cause puisque le processus de recherche d'une règle peut éventuellement produire un résultat différent ;

- ajout ou suppression d'une instruction, autre que l'instruction template : si l'instruction est de type production, le document cible doit être mis à jour. Si c'est une expression qui est ajoutée, supprimée ou modifiée, celle-ci doit être évaluée puis l'instruction associée doit être exécutée.

Les fonctions à mettre en œuvre pour chacun de ces cas sont aussi à mettre en œuvre lors de la modification du document source. Il n'est donc pas nécessaire d'étudier spécifiquement l'édition sur la feuille de transformation.

3.2.3. Fonctions du processeur incrémental

En résumé, pour pouvoir prendre en compte les conséquences de mises à jour, directes ou indirectes, le processeur incrémental doit être en mesure :

- de détecter quelles sont les expressions qui ont besoin d'être réévaluées (cf. section 3.3) ;

- de reconsidérer efficacement les instanciations (cf. section 3.4) ;

- de contrôler les répercussions liées au changement de la valeur d'un paramètre ou de la liste courante de nœuds (cf. section 3.5) ;

- et d'exécuter une instruction de façon incrémentale (cf. section 3.6).

Nous étudions comment réaliser ces différents points dans la suite de cette section.

3.3. Détection des expressions à réévaluer

3.3.1. Formalisation de la modification du document source

Nous avons vu que le résultat d'une expression peut potentiellement être remis en cause suite à la modification du document source (et non de la feuille de transformation). Les modifications élémentaires pouvant survenir sur le document source sont les suivantes :

- sur un élément : ajout et suppression ;
- sur un attribut : ajout, suppression et modification de la valeur ;
- sur un texte : ajout, suppression et modification du contenu.

En reprenant la notation présentée dans la section précédente, une modification du document source peut être modélisée par un tuple $M = (c_m, n_m, p_m)$ avec :

- c_m le chemin de nœuds ascendants à partir duquel le document source a été modifié, jusqu'à la racine du document ;
- n_m le nœud modifié, c'est-à-dire soit inséré comme fils du dernier nœud dans le chemin c_m ou soit supprimé de l'arbre ;
- p_m la position de n_m dans le dernier nœud de c_m.

En utilisant la syntaxe XPath, une modification est un sélecteur de la forme $c_m/n_m[p_m]$ associé au type de la modification (ajout/suppression/modification). Par exemple l'insertion d'une seconde section dans un article est caractérisé par l'expression `/article/section[2]` et de l'opération ajout.

Au niveau du document source, cette modification a des répercutions sur la position des nœuds qui suivent le nœud n_m (les nœuds frères). Cependant, ces répercutions n'ont pas d'influence sur l'exécution de la transformation. En effet, la position p_s d'un nœud est celle dans la liste des nœuds sélectionnés l_s et non pas celle dans l'arbre source (cf. section 2.3). Or cette liste ne change que si le sélecteur qui l'a créée dépend du nœud modifié. La prise en compte du changement de cette liste est étudiée dans la section 3.5. Par conséquent lors de la modification d'un nœud source, il suffit de considérer uniquement le nœud modifié pour détecter les expressions à réévaluer.

3.3.2. Détection des expressions à réévaluer : règles de réévaluation

Le principe de la transformation incrémentale est d'associer aux modifications pouvant survenir sur le document une liste d'expressions à réévaluer. Cependant, il est impossible de considérer *a priori* toutes les modifications sur le document source, ceci à cause du nombre trop important de celles-ci. L'approche choisie est opposée à celle-ci : une analyse de chaque expression est effectuée afin de calculer les modifications sur le document source qui peuvent entraîner une modification du résultat de l'expression. Afin de ne pas énumérer l'ensemble de ces modifications, nous cherchons à calculer un pattern qui permet de les caractériser. À ce pattern est associée une liste d'instructions à réexécuter, correspondant à l'analyse de chaque expression de la feuille de transformation. Ce couple est appelé *règle de réévaluation*. Par exemple, la figure 66 donne la liste des instructions à réévaluer lorsqu'un élément `section` est ajouté dans un élément `article`, caractérisé par le pattern `article/section`.

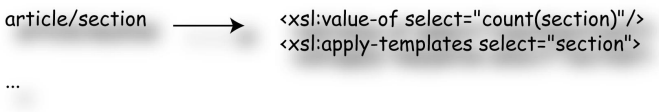


Figure 66. Quelques règles de réévaluation.

Le calcul et l'évaluation du pattern de chaque règle de réévaluation ne peuvent pas s'effectuer pour chaque contexte d'exécution. Ceci pour plusieurs raisons. Premièrement, nous avons vu que la conservation de l'ensemble des contextes d'exécution nécessite beaucoup de ressources mémoire (cf. l'arbre d'exécution section 2.2.5). Or un des objectifs que nous nous sommes fixé est de limiter la taille mémoire requise par l'arbre d'exécution. Une autre raison est qu'il est possible de déterminer dans certains cas quand une expression doit être réévaluée sans la connaissance du contexte dynamique. Par exemple, l'expression `count(section)` de l'instruction `value-of` (cf. figure 66) ne nécessite d'être réévaluée que si un nœud de type `section` est ajouté ou supprimé du document source. Nous proposons donc de calculer les règles de réévaluation sans la connaissance du contexte d'exécution : nous savons donc par avance que le nombre de règles créées sera supérieur au nombre optimum. Ces règles peuvent servir pour filtrer de façon statique les instructions qui n'ont pas besoin d'être réévaluées. Le calcul des règles de réévaluation est donc un prétraitement avant l'exécution incrémentale.

3.3.3. Construction des règles de réévaluation

La liste des règles de réévaluation est calculée en considérant toutes les expressions de la feuille de transformation. Chaque expression est convertie en un ensemble de patterns caractérisant les modifications du document source pouvant modifier le résultat de l'évaluation de l'expression considérée. Pour chaque pattern une règle est créée avec l'instruction correspondante à réexécuter. Puisque la création et l'utilisation de ces règles sont effectuées sans la connaissance du contexte dynamique d'exécution, les patterns ne doivent pas comporter de références vers le contexte dynamique.

Le calcul des règles de réévaluation peut être formalisé comme suit : Soit A l'ensemble des nœuds sélectionnés par l'expression e de l'instruction i avant la modification du document source. Soit B l'ensemble des nœuds sélectionnés par la même expression après la modification du document source. La règle de réévaluation (p, i) est ajoutée si $A \neq B$, avec p le pattern modélisant la modification du document source. Par conséquent, dans la suite nous cherchons à identifier le ou les conditions pour que A et B soient différents en considérant chaque objet syntaxique d'une expression. Pour cela, nous nous reposons sur la fonction *eval-ns* étudiée dans la section 2.3.

Pour pouvoir déterminer que ces ensembles sont différents, il est nécessaire aussi de déterminer quand l'évaluation d'un prédicat change suite à la modification du document source. Nous étudions donc aussi les changements de l'évaluation de la fonction *eval-bo*.

Pour alléger la suite de ce chapitre, nous considérons uniquement l'opération d'ajout d'un nouveau nœud noté n_m dans l'arbre source. Le travail effectué ci-dessous peut facilement être généralisé pour les autres opérations d'édition, y compris pour l'opération de suppression. Pour cette dernière, la détection des expressions à réévaluer doit être réalisée avant l'opération de suppression.

3.3.3.1 Objets primitifs, opérateurs et fonctions

Les objets primitifs, les opérateurs et les fonctions statiques ne produisent pas de règles puisqu'ils ne dépendent pas du contexte source pour être évalués. Les fonctions dynamiques qui utilisent le contexte source, c'est-à-dire la fonction position() et la fonction last(), doivent être réévaluées lorsque la liste courante de nœuds ou la position courante changent. Ce changement est étudié dans la section 3.5.

3.3.3.2 Étape sans prédicat

Nous rappelons que l'évaluation d'une étape d'expression XPath sans prédicat est représentée par l'équation sémantique suivante :

$$A = eval\text{-}ns[a::nt]\rho = \{ n \in t_\Sigma / n \in axis[a]n_\rho \wedge NodeTest[nt]n \}$$
$$B = eval\text{-}ns[a::nt]\rho = \{ n \in (t_\Sigma \cup n_m) / n \in axis[a]n_\rho \wedge NodeTest[nt]n \}$$

Pour que ces deux ensembles soient différents, il est nécessaire que le nouveau nœud source respecte la condition pour appartenir à l'ensemble des nœuds sélectionnés. Par conséquent :

$$A \neq B \Leftrightarrow n_m \in axis[a]n_\rho \wedge NodeTest[nt]n_m$$

La condition d'appartenance de n_m à l'ensemble des nœuds résultant de la fonction *axis* ne peut pas être évaluée statiquement à cause du nœud source courant n_ρ qui est inconnu. Par contre cette condition permet de calculer les nœuds source courant pour lesquels l'expression doit être réévaluée. Pour résumer, une expression de type *a::nt* se transforme en :

Une règle de réévaluation : *nt*
Un paramètre dynamique : $n_s \in axis^{-1}[a]n_m$

Par exemple, l'expression descendant::section doit être réévaluée lorsqu'un élément du même nom (section) est ajouté ou supprimé de l'arbre source. De plus, elle doit être réévaluée uniquement pour les nœuds source tels que $n_s \in axis[ancestor]n_m$ où n_m est le nœud de type section.

Lorsque le résultat d'une sélection est converti vers le type chaîne de caractères, l'ensemble des descendants de type texte du premier nœud sélectionné est extrait lors de cette conversion. Par conséquent, il est nécessaire d'ajouter un pattern qui reflète la modification d'un de ces nœuds texte :

Une règle de réévaluation : *nt/descendant::text()*

3.3.3.3 Étape avec prédicat

L'évaluation d'une étape avec prédicat est représentée par l'équation sémantique suivante :

$$A = eval\text{-}ns[a :: nt[e]]\rho = \{ n \in t_\Sigma / n \in axis[a]n_\rho \wedge NodeTest[nt]n \wedge eval\text{-}bo[e]\rho_2 \}$$
$$B = eval\text{-}ns[a :: nt[e]]\rho = \{ n \in (t_\Sigma \cup n_m) / n \in axis[a]n_\rho \wedge NodeTest[nt]n \wedge eval\text{-}bo[e]\rho_2 \}$$

avec $n_{\rho 2} = n$ et $l_{\rho 2} = eval[a :: nt[e]]\rho$

Comme pour le cas précédent, les ensembles A et B sont différents si la condition pour appartenir à B est vérifiée pour le nouveau nœud. Cependant, ces ensembles peuvent aussi être différents si la condition sur les autres nœuds change. De façon formelle :

$$
\begin{aligned}
A \neq B \Leftrightarrow & \\
& \text{NodeTest}[nt]n_m \wedge n_m \in \text{axis}[a]n_\rho \wedge \text{eval-bo}[e]\rho_2 \quad (1)\\
& \vee (\exists\, n \in A \,/\, \text{eval-bo}_A[e]\rho_3 \neq \text{eval-bo}_B[e]\rho_3) \quad (2)\\
& \qquad \text{avec } n_{\rho 2} = n_m \text{ et } l_{\rho 2} = \textit{eval-ns}[a :: nt]\rho \\
& \qquad \text{et } n_{\rho 3} = n \text{ et } l_{\rho 3} = \textit{eval-ns}[a :: nt]\rho
\end{aligned}
$$

La question est de savoir s'il est possible d'évaluer la condition statiquement. La première condition (1) est similaire au cas précédent sauf pour l'évaluation du prédicat e. Pour être évalué, ce prédicat dépend du contexte ρ_2 composé de $n_{\rho 2}$, qui est connu puisque c'est le nœud modifié, et de $l_{\rho 2}$ qui peut être calculé. En effet, pour cela il suffit de connaître n_ρ qui est calculé comme suit :

$$n_m \in \text{axis}[a]n_\rho \Rightarrow n_\rho \in \text{axis}^{-1}[a]n_m$$

De cette condition, il est possible de déduire une règle de réévaluation et un paramètre du contexte dynamique :

Une règle de réévaluation : $nt[e]$
Un paramètre dynamique : $n_\rho \in axis^{-1}[a]n_m$

La seconde condition repose sur l'évaluation booléenne du prédicat pour l'ensemble des nœuds source instanciés contenus dans A. Deux cas sont à considérer selon les deux équations suivantes :

$$
\begin{aligned}
& \textit{eval-bo}_A[a::nt]\rho_3 \neq \text{eval-}bo_B[a::nt]\rho_3 \\
& \qquad \Leftrightarrow \exists\, n \in axis[a]n_\rho \wedge NodeTest[nt]n \\
& \textit{eval-bo}_A[position() = i]\rho_3 \neq \textit{eval-bo}_B[position() = i]\rho_3 \\
& \qquad \Leftrightarrow \forall\, n \in l_{\rho 3} \,/\, p_{\rho 3} \text{ de } n \text{ dans } A \neq p_{\rho 3} \text{ de } n \text{ dans } B
\end{aligned}
$$

Lorsque le prédicat e a la forme $a::nt$ alors A et B sont différents pour les mêmes raisons évoquées dans la section précédente. La règle de réévaluation nt est alors créée (comme précédemment).

Lorsque le prédicat e à la forme $position() = i$ alors il est nécessaire de réévaluer les nœuds de $l_{\rho 3}$ qui ont une position différente après la modification, c'est-à-dire les nœuds qui ont été décalés par l'insertion (dans ce cas) de n_m. La règle $\text{nt}[\text{position}() < p_{n_m})]$ est alors ajoutée.

Par exemple, prenons l'expression `section[position() = 4]` qui sélectionne l'élément `section` ayant une position égale à quatre. L'application des règles ci-dessus montre que cette expression doit être réévaluée selon le pattern suivant : `section[position() < 4].` L'évaluation de cette expression peut changer si une section est ajoutée ou supprimée avant la position quatre.

3.3.3.4 Enchaînement d'étapes : chemin de localisation

L'évaluation d'un chemin de localisation est représentée par l'équation sémantique suivante

$$\textit{Soit } A = \textit{eval}[lp_1/lp_2]\rho = \{\, n_2 \in t_\Sigma / n_1 \in \text{eval-ns}[lp_1]\rho,\ n_2 \in \textit{eval-ns}[lp_2]\rho_1 \,\}$$
$$\text{avec } l_{\rho 1} = \textit{eval-ns}[lp_1]\rho \text{ et } n_{\rho 1} \in l_{\rho 1}$$

Cette équation indique que l'évaluation d'un chemin de localisation dépend à la fois de la partie droite lp_1 et de la partie gauche lp_2. Par conséquent, les patterns sont créés pour chacune de ces parties selon les règles décrites dans les sections précédentes.

3.3.4. Bilan

Lors de la création des patterns à partir des conditions pour changer l'évaluation d'une expression, nous avons vu que la caractérisation des modifications du document source est relâchée. Par conséquent, trop d'expressions sont détectées comme étant à réévaluer alors que ce n'est pas nécessairement le cas. Le nombre d'instructions réexécutées est donc supérieur au nombre optimum. Il faut noter cependant que toutes les instructions qui doivent être réexécutées sont, quant à elles, détectées.

Plusieurs optimisations peuvent être envisagées pour raffiner la détection des expressions à réévaluer, en particulier la prise en compte du contexte de déclaration d'une expression. Deux niveaux sont identifiés :

- le contexte dans un prédicat : lors de la conversion d'un prédicat, les sélecteurs de ce prédicat sont évalués à partir du nœud de l'étape du prédicat. Par conséquent, il est possible de préfixer le pattern construit à partir de ces sélecteurs par la partie gauche de l'étape du prédicat. Par exemple, l'expression `section[title]` doit être réévaluée notamment pour les nœuds respectant le pattern `section/title` (au lieu du pattern `title` calculé dans la section 3.3.3.3) ;

- le contexte à l'intérieur d'une règle : chaque expression relative est évaluée à partir d'un nœud qui respecte le pattern de la règle qui contient l'expression. Par conséquent, l'ajout du pattern de la règle comme préfixe à chaque pattern des règles de réévaluations créées par les expressions de cette règle les rend plus précis. Par exemple, l'expression `section` de l'instruction ligne 18 qui est déclarée dans la règle sélectionnant les nœuds de types `article` est convertie vers le pattern `article/section` (au lieu du pattern `section`).

3.4. Remise en cause des instanciations

3.4.1. Principe

Durant l'exécution de l'instruction `apply-templates` (et `apply-imports`), une règle est recherchée pour chacun des nœuds sélectionnés par l'expression associée à cette instruction. Lorsque le document source est modifié, cette recherche peut produire un résultat différent (cf. section 2.2.3).

De façon générale, le processus d'instanciation doit considérer toutes les règles définies dans la feuille de transformation. Par conséquent, chaque fois que le document source est modifié, toutes les précédentes instanciations doivent être considérées. Pour éviter ce surcoût, nous prenons en compte le fait que la plupart du temps, seul un sous-ensemble des règles peut être instancié pour une instruction `apply-templates` donnée. À partir de ces dépendances `apply-templates/template`, le processus d'instanciation pour une instruction `apply-templates`

donnée peut être limité à un sous-ensemble des règles. Par exemple, considérons les règles de transformation suivantes :

```
1.  <xsl:template match="article"> … </xsl:template>
2.  <xsl:template match="section"> … </xsl:template>
3.  <xsl:template match="author"> … </xsl:template>
```

L'exécution de l'instruction <xsl:apply-templates select="section"/> ne peut conduire qu'à l'instanciation de la règle ligne 2 (en dehors de la règle de transformation par défaut qui sélectionne tous les éléments), puisque seuls des éléments de type section peuvent être sélectionnés par l'instruction apply-templates.

3.4.2. Construction du graphe de dépendance apply-templates/template

Formellement, le problème de calculer les dépendances entre l'instruction apply-templates et l'instruction template peut se formuler comme suit :

Soit N_A la liste de noeuds que l'instruction apply-templates A peut sélectionner grâce à l'expression spécifiée dans l'attribut select. Soit N_T l'ensemble des noeuds que la règle T peut instancier. La règle T est dite dépendante de l'instruction A si est seulement s'il existe au moins un nœud n appartenant à N_A et qui appartient aussi à N_T. Formellement :

$$T \text{ dépends de } A \Leftrightarrow N_A \cap N_T \neq \varnothing \Leftrightarrow A \text{ est assorti à } T \qquad \text{(Eq 1)}$$

Concrètement, la façon de déterminer ces dépendances est d'effectuer une comparaison (pattern matching) entre une expression (et non un nœud) du apply-templates et un pattern d'une règle. Comme cette fonction est différente de la fonction *match*, nous en introduisons une nouvelle appelée *match-p*. Cette fonction calcule l'assortiment entre une expression et un pattern. Dans la suite nous donnons les équations sémantiques de base permettant de calculer ces dépendances :

$$\begin{aligned}
&\textit{match-p} : \textit{Expr} \rightarrow \textit{Pattern} \rightarrow \textit{Boolean}\\
&\textit{match-p}[q_1][q_2] = \textit{vrai si } q_1 = q_2, \textit{ faux sinon}\\
&\textit{match-p}[*][q] = \textit{vrai}\\
&\textit{match-p}[\textit{node}()][q] = \textit{vrai}\\
&\textit{match-p}[\textit{text}()][q] = \textit{faux}\\
&\textit{match-p}[q][*] = \textit{vrai}\\
&\textit{match-p}[*][*] = \textit{vrai}\\
&\textit{match-p}[\textit{node}()][*] = \textit{vrai}\\
&\textit{match-p}[\textit{text}()][*] = \textit{faux}
\end{aligned}$$

La première équation signifie qu'une expression q1 correspond à un pattern q2 uniquement si leur valeur est identique.

À partir de ces équations, nous pouvons établir les deux équations suivantes :

$$\begin{aligned}
&\textit{match-p}[\textit{child} :: nt_A][\textit{child} :: nt_T] =\\
&\qquad \textit{vrai si match-p}[nt_A][nt_T], \textit{ faux sinon}\\
&\textit{match-p}[\textit{parent} :: nt_A][\textit{child} :: nt_T] =\\
&\qquad \textit{vrai si match-p}[nt_A][nt_T], \textit{ faux sinon}\\
&\textit{match-p}[\textit{descendant} :: nt_A][\textit{child} :: nt_T] =
\end{aligned}$$

$$\text{vrai si match-p}[nt_A][nt_T]\text{, faux sinon}$$
$$\text{match-p}[\text{ancestor :: } nt_A][\text{child :: } nt_T] =$$
$$\text{vrai si match-p}[nt_A][nt_T]\text{, faux sinon}$$

Preuve

Pour effectuer la preuve de ces équations, nous introduisons une nouvelle fonction ***eval-s*** (s=statique) qui dénote l'évaluation d'une expression de façon statique sans contexte dynamique. Elle retourne l'ensemble des nœuds que l'expression peut potentiellement sélectionner. Cette fonction et sa sémantique sont décrits comme suit :

$$\text{eval-s} : Expr \rightarrow Set(Node)$$
$$\text{eval-s}[a :: nt] = \{ n / m \in t_\Sigma, n \in Axis[a]m \wedge NodeTest[nt]n \}$$

La première équation est vraie si

$$\text{eval-s}[\text{child :: } nt_A] \cap \text{eval-s}[\text{child :: } nt_T] \neq \varnothing \qquad \text{(d'après Eq 1)}$$
$$\Leftrightarrow \{ n / m \in t_\Sigma, n \in Axis[child]m \wedge NodeTest[nt_A]n \}$$
$$\cap \{ n / m \in t_\Sigma, n \in Axis[child]m \wedge NodeTest[nt_T]n \} \neq \varnothing$$
$$\Leftrightarrow \exists n \in t_\Sigma, NodeTest[nt_A]n = NodeTest[nt_T]n$$
$$\Leftrightarrow \text{match-p}[nt_A][nt_T]$$

Les autres équations se démontrent de la même façon.

La dernière équation que nous donnons porte sur les chemins de localisation :

$$\text{match-p}[lp_{1A} / lp_{2A}][lp_{1T} / lp_{2T}] = \text{vrai si match-p}[lp_{1A}][lp_{1T}] \wedge \text{match-p}[lp_{2A}][lp_{2T}]$$

Preuve

La fonction ***eval-s*** est étendue par l'équation suivante :

$$\text{eval-s}[\, lp_1 / lp_2 \,] =$$
$$\{ n / m \in t_\Sigma, n_1 \in \text{eval-ns}[lp_1]m, n \in \text{eval-ns}[lp_2]n1 \}$$

D'où la preuve suivante :

$$\text{eval-s}[lp1A / lp2A \,] \cap \text{eval-s}[lp1T / lp2T] \neq \varnothing$$
$$\Leftrightarrow \{n / m \in t_\Sigma, n_1 \in \text{eval-ns}[lp_{1A}]m, n \in \text{eval-ns}[lp_{2A}]n1 \}$$
$$\cap \{ n / m \in t_\Sigma, n_1 \in \text{eval-ns}[lp_{1T}]m, n \in \text{eval-ns}[lp_{2T}]n_1 \} \neq \varnothing$$
$$\Leftrightarrow \exists n / \exists n_1 \in \text{eval-s}[lp_{1A}][lp_{1T}],$$
$$n \in \text{eval-ns}[lp_{2A}]n_1 \wedge n \in \text{eval-ns}[lp_{2T}]n_1$$
$$\Leftrightarrow \text{match-p}[lp_{1A}][lp_{1T}] \wedge \text{match-p}[lp_{2A}][lp_{1T}]$$

Par exemple, l'expression `section/title` ne correspond pas au pattern `article/title` car la fonction ***match-p[section][article]*** est évaluée à faux. L'expression `section/*` correspond au pattern `section/node()` car la fonction ***match-p[section][section]*** est vraie ainsi que la fonction ***match-p[*][node()]***.

3.4.3. Application à l'exemple

La figure 67 montre le graphe de dépendance pour la feuille de transformation donnée en annexe A, dont un extrait est donné ci-dessous :

```
8.    <xsl:template match="article">
…
18.             <xsl:apply-templates select="section">
…
24.           <td><xsl:apply-templates select="artheader/authorgroup"/></td>
…
33.     <xsl:template match="section">
…
51.     <xsl:number value="position()" format="1."/>
…
57.           <xsl:apply-templates select="section">
…
64.     <xsl:template match="authorgroup">
```

Une partie des règles par défaut, ainsi que les instructions apply-templates associées, est représentée. Nous avons aussi ajouté la règle <template match="section[title]"> pour montrer la dépendance multiple de l'instruction apply-templates vers une règle.

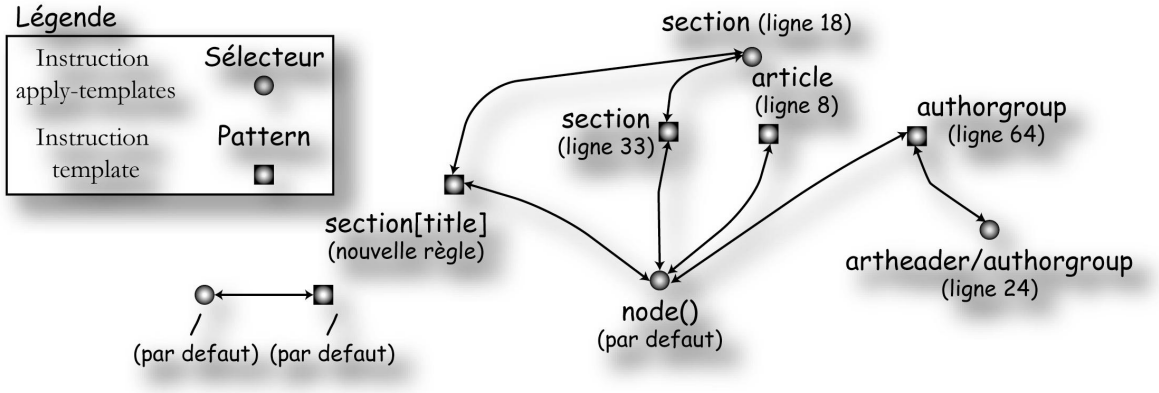


Figure 67. Graphe de dépendance pour la feuille de transformation donnée en annexe A.

Ce graphe montre notamment que l'expression section de l'instruction le ligne 18 ne peut instancier que deux règles : la règle ligne 33 dont le pattern est section et la nouvelle règle dont le pattern est section[title]. La règle qui sera réellement instanciée dépend du contexte du nœud source traité (s'il contient un titre ou non). Lors du processus d'instanciation, seules ces deux règles sont testées, sur les sept existantes (deux règles par défaut ne sont pas montrées).

3.5. Contrôle des répercussions

Dans certains cas, une expression doit être réévaluée suite à une modification indirecte du contexte d'exécution. Par exemple, lorsque l'évaluation de l'expression de l'instruction apply-templates ligne 18 change, alors l'expression position() ligne 51 peut changer. Par conséquent les patterns associés à la première expression doivent aussi être associés à la seconde expression. La difficulté est que ces expressions n'appartiennent pas à la même règle de transformation.

La première solution est d'utiliser le graphe de dépendance apply-templates/template calculé dans la section précédente. En effet, ce graphe permet de connaître de façon statique les liens entre deux expressions appartenant à des règles différentes. Par exemple, les patterns

à associer à l'instruction ligne 51, qui appartiennent à la règle section ligne 33, sont ceux associés aux instructions apply-templates dépendantes de cette règle, c'est-à-dire l'instruction ligne 18 et l'instruction par défaut avec l'expression node(). Il faut noter que la portée du contexte d'exécution établi par le processus d'instanciation est locale à la règle (cf. section 2.2.3). Il suffit donc de rechercher les instructions apply-templates directement liées à la règle traitée. Le problème avec cette solution est que les règles de réévaluation sont approximatives : trop d'expressions peuvent être marquées à réévaluer alors qu'elles ne le devraient pas.

La seconde solution est de répercuter les modifications au moment de l'exécution de la transformation incrémentale. À ce moment, lorsqu'une instruction apply-templates est exécutée et que la position d'un nœud sélectionné a changé, alors un drapeau est levé à vrai pour indiquer que les expressions faisant référence à cette position sont à réévaluer. Cette solution suppose que le mode d'exécution s'effectue selon le modèle de parcours de l'arbre d'exécution (cf. section suivante).

3.6. Exécution incrémentale

Grâce à la liste des règles de réévaluation calculée dans la section 3.3, les instructions qui nécessitent d'être réexécutées sont maintenant connues. Il suffit pour cela d'appliquer une correspondance entre les patterns de ces règles et le nœud qui est en train d'être modifié. Seules les instructions associées aux patterns correspondants nécessitent d'être réexécutées. Ensuite, deux modèles d'exécution de ces instructions peuvent être envisagés. Dans le premier modèle, le processeur exécute la transformation en parcourant l'arbre d'exécution. Dans le second modèle, chaque instruction de la liste des patterns identifiés est directement exécutée. Les deux modèles ainsi que les algorithmes associés sont décrits ci-dessous. Nous précisons également pour chacune de ces méthodes quelles informations sont conservées dans l'arbre d'exécution. Dans la dernière sous-section, nous comparons ces deux modèles.

3.6.1. Parcours de l'arbre d'exécution

Dans ce modèle, un parcours en profondeur d'abord de l'arbre d'exécution est effectué. Ce parcours est similaire à une approche non incrémentale, c'est-à-dire que les variables et les paramètres sont d'abord empilées au début de leur parcours puis ensuite dépilées à la fin de leur parcours, le contexte cible est mis à jour, etc. Dans le cadre d'un processeur incrémental, les actions particulières suivantes sont effectuées selon le type du nœud d'exécution :

- **Contrôle.** Si l'instruction a besoin d'être réexécutée, alors elle est exécutée de façon incrémentale. La figure 68 ci-dessous donne l'algorithme pour l'instruction apply-templates ;

- **Production.** Si l'instruction a besoin d'être réexécutée, alors elle est exécutée de façon incrémentale : pour les instructions qui produisent des caractères, ceux générés lors de la précédente transformation sont remplacés. Sinon, seul le contexte cible est mis à jour : pour les instructions qui produisent des caractères, le compteur de caractères est augmenté du nombre de caractères générés avant la modification. Pour les instructions qui génèrent un élément, le compteur de caractères est réinitialisé et l'élément cible correspondant devient le nœud cible courant (n_c) ;

Lorsqu'une instruction doit être exécutée, son algorithme incrémental est utilisé. La figure 68 donne l'algorithme qui permet l'exécution incrémentale de l'instruction apply-templates.

```
1. nodeList <- Evaluate select expression
2. If ( sort instruction specified )
3.     sort( nodeList )
4. End If
5. For each Node in NodeList
6.     If ( Node ∉ previousNodeList )
7.         // The source node was not selected during the previous transformation
8.         Execute apply-templates instruction with Node as context
9.     Else
10.            // Test if the template matching has changed
11.            template <- findTemplate( Node )
12.            If ( template = previousTemplate( Node ) )
13.                // Same template
14.                If ( not Same position as previously )
15.                    // The generation order has changed
16.                    changeTargetPosition()
17.                End If
18.                Incrementally execute children with Node as context
19.            Else
20.                // Not the same template
21.                Destroy previously generated target
22.                Execute apply-templates instruction with Node as context
23.            End If
24.        End
25.    End For
```

Figure 68. Algorithme incrémental pour l'instruction apply-templates.

L'algorithme pour l'instruction for-each est similaire, excepté qu'il n'est pas nécessaire de rechercher une règle. Par conséquent, le test qui vérifie si les instanciations n'ont pas changées n'est pas exécuté. Pour l'instruction apply-imports, c'est le contraire, seul ce test est exécuté. Pour l'instruction value-of, les nouveaux caractères remplacent ceux générés précédemment.

3.6.2. Exécution directe des instructions

Dans le second modèle, les instructions qui nécessitent d'être réévaluées sont exécutées une par une. Puisque chaque instruction peut avoir plusieurs exécutions, les nœuds d'exécution correspondant doivent être recherchés. Ensuite, pour chaque nœud, le contexte d'exécution minimal est calculé comme suit :

- le nœud source courant n_s est facilement restauré car il est porté par le nœud représentant l'exécution d'une instruction template ;
- la position courante p_s est donnée par la position du nœud d'exécution à l'intérieur du nœud parent ;
- la liste des fils de ce nœud parent constitue la liste courante l_s ;
- la liste des variables qui nécessitent d'être calculées est donnée par les références contenues dans l'expression de l'instruction. Ces variables sont calculées lors de l'évaluation de l'expression.

Pour restaurer le contexte cible, une vue particulière de l'arbre d'exécution est définie. Cette vue montre uniquement les instructions de production. La notion de vue est détaillée dans la spécification DOM Level 2 [DOM 00].

Pour ce modèle, les algorithmes incrémentaux sont similaires à ceux présentés dans la section précédente. La seule différence est que le parcours de l'arbre n'est pas effectué.

L'application de ce processus nécessite de restaurer le contexte d'exécution pour chacune des instructions à réexécuter. En particulier, la valeur d'une même variable pour un même contexte peut, dans ce modèle, être calculée plusieurs fois. Une optimisation simple est de trier les nœuds d'exécution dans le but d'éviter ces calculs redondants. Cela permet de conserver la valeur des variables qui sont dans la portée du nœud suivant à exécuter. La seconde raison pour trier les nœuds d'exécution est que, dans certains cas, la génération des nœuds cible doit être effectuée dans l'ordre de l'arbre d'exécution. Par exemple, pour les documents comportant des références croisées, il est parfois requis que l'identificateur soit déclaré avant d'être utilisé.

3.6.3. Comparaison des modèles

Le principal inconvénient du premier modèle est que le parcours de l'arbre d'exécution est proportionnel à la taille de l'arbre cible. Par conséquent, pour de gros documents, le temps de mise à jour peut être important. Dans le second modèle, avant qu'une instruction puisse être exécutée, il est nécessaire d'effectuer plusieurs calculs pour restaurer le contexte. De plus, dans le but d'augmenter les performances, les liens entre les instructions et les nœuds d'exécution doivent être conservés. Les nœuds d'exécution de production nécessitent aussi d'être stockés afin de pouvoir restaurer le contexte cible. Par conséquent, le second modèle est plus coûteux en taille mémoire.

En résumé, lorsque peu d'instructions nécessitent d'être réévaluées, le second modèle semble plus adapté. Néanmoins, dans le contexte de l'édition, le second modèle permet de réexécuter en priorité les objets graphiques qui sont au centre de l'intérêt de l'auteur. L'optimisation qui consiste à trier les nœuds d'exécution peut même être omise en considérant que le temps global pour mettre à jour le document cible dans sa totalité est de moindre importance que la mise à jour des éléments ayant une priorité élevée.

3.7. Bilan

Dans cette section, nous avons décrit nos solutions pour mettre à jour le document résultat d'une transformation après l'édition du document source. Ces solutions permettent aussi de prendre en compte les modifications de la spécification de la transformation.

Les solutions proposées ont été appliquées pour un sous-ensemble du langage d'expression XPath. En particulier, les variables n'ont pas été traitées. Cependant, nous avons défini un cadre théorique suffisamment générique pour compléter facilement les résultats décrits dans cette section. On peut noter que le traitement des variables est équivalent au traitement sur le paramètre de la liste courante l_s qui partage les mêmes propriétés concernant l'usage et la portée.

4. Évaluation et expérimentation

Les techniques présentées dans les sections précédentes permettent l'implantation de processeurs de transformation incrémentaux. L'objectif de cette section est de présenter le processeur incrémental que nous avons réalisé. Nous décrivons ensuite les différentes évaluations que nous avons menées. Nous terminons par décrire son intégration dans notre outil d'édition présenté dans le chapitre précédent.

4.1. Implantation

Ces techniques ont été partiellement intégrées dans Xalan [Xalan], le processeur de transformation XSLT développé au sein de la fondation Apache. Chacune des fonctions identifiées dans la section 3.2.3 ont été implantées avec les restrictions suivantes :

- la détection des expressions à réévaluer a été implantée pour les éléments syntaxiques considérés dans la section 3.3.3. Les expressions sortant de ce cadre syntaxique sont réévaluées systématiquement.
- la remise en cause des instanciations s'effectue systématiquement. La construction du graphe de dépendance `apply-templates/template` a été implantée pour montrer la validité des règles de construction. Par contre ce graphe n'a pas été connecté avec le processus d'instanciation ;
- le contrôle des répercussions s'effectue selon la seconde solution, c'est-à-dire pendant le parcours de l'arbre d'exécution. La première solution n'a pas été implantée ;
- l'exécution incrémental respecte la première méthode du parcours de l'arbre. La quasi-totalité des algorithmes incrémentaux associés à chacune des instructions XSLT a été implantée.

En terme de développement, ce travail représente 10000 lignes de code Java ajoutés aux 140000 lignes existantes dans la version 2.0.0 de Xalan.

4.2. Évaluation

Dans le but d'évaluer notre implantation courante du processeur incrémental, quelques mesures ont été effectuées. Ces mesures évaluent le temps nécessaire pour mettre à jour le résultat de la transformation et l'espace mémoire requis. Le résultat de ces évaluations est résumé dans le tableau 4, 5 et 6.

Concernant la vitesse d'exécution, nous avons mesuré le temps pris pour mettre à jour le résultat à partir de quelques opérations d'édition, comme l'ajout d'un auteur, d'une section et d'un titre d'article dans le document d'exemple (cf. section 2.2.1). Les résultats obtenus sont résumés dans le tableau 4 et 5. Le tableau 4 donne les mesures effectuées sur un document de type Docbook en utilisant les feuilles de transformation écrites par Norman Walsh qui produisent des documents html. Ces feuilles contiennent 1236 règles et ne sont pas optimisée pour une transformation incrémentale. Elles contiennent beaucoup d'expressions génériques (`*`, `node()`, etc.), de variables et de paramètres. Par conséquent, le nombre d'instructions à réexécuter est largement surestimé. Pour changer le titre de l'article, 795 instructions sont détectées comme étant à réévaluer alors que théoriquement une seule instruction est nécessaire. De plus, beaucoup de temps est gaspillé à cause de l'évaluation systématique des

variables durant le parcours de l'arbre, ce qui représente presque 100% du temps du processus incrémental. Le résultat est que la plupart des modifications reste compris dans un temps raisonnable.

Une seconde évaluation a été effectuée sur une feuille de transformation plus petite (50 règles). Cette feuille, qui est une extension de la feuille présentée en annexe A, a été appliquée sur les mêmes documents source. Elle contient peu de variables et aucune expression générique. Par conséquent, les expressions qui nécessitent d'être réévaluées sont déterminées de façon plus précise que celles contenues dans les feuilles de Norman Walsh.

	Première transformation (avant édition)		**Opération d'édition nulle**		**Change le titre de l'article**		**Insertion d'une section**	
Nombre d'instructions à réexécuter	N/A		0		795		819	
Temps pour trouver les instructions à réexécuter	N/A		0		80ms		80ms	
Temps de l'exécution incrémentale	4,5s		2,8s		2s		2,1s	
Temps complet / ratio	4,5s	1.0	2,8s	0.62	2,8s	0.62	2,9s	0.64

Tableau 4. Coût d'une transformation appliquée sur les feuilles de transformation Docbook de Norman Walsh

	Première transformation (avant édition)		Opération d'édition nulle		Ajout d'un auteur		Insertion d'une section	
Nombre d'instructions à réexécuter	N/A		0		18		20	
Temps pour trouver les instructions à réexécuter	N/A		0		<10ms		<10ms	
Temps de l'exécution incrémentale	2,3s		0,1s		0,2s		0,2s	
Temps complet / ratio	2,3s	N/A	0,1s	0.04	0,2s	0.08	0,3s	0.13

Tableau 5. Temps d'une transformation appliquée sur une version étendue de la feuille de transformation en annexe A.

Le coût en mémoire n'est pas non plus prohibitif. Le surcoût mesuré est surtout du à l'arbre d'exécution qui représente une fraction de la mémoire occupée par le document source et la feuille de transformation.

	Document A	Document B
Document source (Kbytes)	193	505
Document cible (Kbytes)	224	596
Taille de l'arbre d'exécution (sans les nœuds de production) (Kbytes)	106	259
Taille de l'arbre d'exécution (avec les noeuds de production) (Kbytes)	153	364
Ratio (doc. source + doc. target) / execution	36%	33%

Tableau 6. Mémoire additionnelle utilisée par le processeur de transformation incrémental (feuilles de transformation de Norman Walsh).

4.3. Expérimentation

Ce processeur incrémental a été intégré dans le système d'édition décrit dans le chapitre précédent. Pour cela, nous avons dû effectuer les tâches suivantes, en plus de l'implantation de techniques incrémentales dans le processeur :

- l'implantation des événements « incrémentaux » produits par le processeur incrémental. En effet, Xalan permet de produire le document cible par événements, à travers l'interface SAX (cf. chapitre 2 section 2.1.2). Pour implanter notre processeur incrémental, nous avons étendu cette interface par des fonctions comme `startElement` pour parcourir l'élément cible courant, ou `replaceCharacter` pour remplacer les caractères précédemment générés par de nouveaux ;

- l'implantation de l'interface DOM au sein de Xalan. Pour permettre l'édition des feuilles de transformation, nous avons implanté les méthodes de DOM permettant la modification d'un arbre XML, comme les méthodes `appendChild` et `addAttribute`. Ces méthodes n'étaient pas implantées dans la version non incrémentale de Xalan.

Grâce à ce processeur incrémental, notre éditeur permet l'édition du document source et des feuilles de transformation de façon suffisamment réactive.

5. Conclusion

Dans ce chapitre nous avons décrit de façon formelle la sémantique du langage XPath. Une première tentative de modélisation formelle du langage XPath a été effectuée par P. Walder [Wadler 00]. Par rapport à la modélisation qu'il décrit, nous sommes allé plus loin notamment en considérant la liste courante de nœuds et la position courante du contexte d'évaluation. Notre formalisation est donc plus complète et plus proche de la version finale de XPath. En effet, P. Walder l'a fait partiellement parce que la spécification était en cours de définition. Notamment, la notion d'axe n'existait pas, seules quelques fonctions (`ancestor(e)`, `ancestor-or-self(e)` et `parent(e)`) permettait de sélectionner les nœuds ascendants. Les patterns ne pouvaient pas contenir de prédicat.

Dans cette thèse, nous avons appliqué la transformation incrémentale dans le cadre de l'édition de documents. Il existe d'autres champs d'application de la transformation incrémentale que nous citons ci-dessous :

- l'optimisation de la première transformation : certaines techniques présentées dans ce chapitre peuvent être utilisées pour améliorer la transformation initiale, en particulier le graphe de dépendances décrit dans la section 3.4 ;

- la mise à jour de sites Web : les sites Web créés à partir de transformations XSLT nécessitent d'être mis à jour lorsque des données sources sont modifiées. Étant donné qu'un site peut contenir un nombre important de pages, sa mise à jour est d'autant plus rapide si elle est incrémentale ;

- le changement des paramètres de la transformation : l'interprétation des feuilles de transformation nécessite parfois la connaissance de paramètres externes. Notamment, ces paramètres peuvent être utilisés pour générer des présentations adaptées dynamiquement au contexte de restitution, comme nous l'avons montré dans le chapitre 4. Le processeur incrémental permet de prendre en compte de façon optimal le

changement de tels paramètres et fournit ainsi un service d'adaptation dynamique plus adapté en terme de temps de réponse.

Pour terminer, le temps de mise à jour du résultat de la transformation peut être relativement important, même avec une mise à jour incrémentale optimal. Il existe des cas où l'édition du document source (ou de la feuille de transformation) entraîne une remise en cause complète du résultat. C'est le cas par exemple lorsque l'auteur change la langue du document. Le plus important néanmoins est que la mise à jour soit réalisée de la façon la plus rapide possible.

Chapitre 7

Conclusion et perspectives

1. Rappel des objectifs

Ce mémoire a traité des besoins d'édition et de présentation pour des documents intégrant de multiples médias. La diffusion massive de tels documents, dits multimédias, est devenue aujourd'hui une réalité, et ce sur de multiples supports, malgré les coûts de production qui restent encore très élevés. Dans la perspective d'améliorer cette diffusion, nous nous sommes fixé comme objectifs de contribuer à la fois sur :

- la réduction du coût de production des documents multimédias, tout en augmentant leur qualité et leur pérennité ;
- l'élaboration d'une méthode permettant l'adaptation de présentations multimédias aux contextes de l'utilisateur.

2. Démarche de travail et bilan théorique

Pour atteindre ces objectifs, nous avons étudié les concepts fondamentaux du génie documentaire. Cette étude montre en particulier que la modélisation est au cœur de toutes les applications documentaires et que c'est grâce à elle que les traitements peuvent être réutilisés. La réduction des coûts de production de documents multimédias passe donc forcement par une modélisation formelle de ces documents. Or, nous avons montré l'incapacité des langages de modélisation existants pour représenter formellement les documents multimédias, ceci malgré l'effort important déployé dans la définition de tels langages. Par ailleurs, nous avons analysé les systèmes existants permettant de produire des présentations adaptées. Nous avons montré les limitations de ces systèmes grâce à une synthèse sur l'ensemble des techniques d'adaptation utilisées. Ces limitations se traduisent par un niveau d'adaptation insuffisant.

Le premier résultat de cette thèse est l'élaboration d'une méthode permettant de produire des présentations multimédias à partir de documents respectant un modèle métier. Puisque la sémantique de présentation ne peut pas être décrite formellement par ce modèle, nous avons été amené à séparer les informations de contenu des informations de présentation dans le cadre de la modélisation de classe de documents multimédias. Ce constat nous a conduit à analyser les techniques d'assemblage de ces deux ensembles d'informations. Le résultat de cette analyse s'est traduit par la nécessité d'utiliser une technique de transformation. De plus, nous avons montré à travers plusieurs expérimentations les limitations d'une description directe de la présentation. Nous avons alors cherché à capturer au mieux les intentions de l'auteur à travers un vocabulaire de haut niveau. Dans ce cadre, l'approche fondée sur les transformations a montré sa force pour générer une présentation à partir de ce vocabulaire.

Le second résultat est la conception d'une architecture permettant de produire des présentations adaptées au contexte utilisateur. Nous avons proposé trois composants : la transformation, le formatage et l'exécution. Pour chacun de ces composants, nous avons

identifié leur rôle précis dans un système adaptable et les techniques d'adaptation qu'ils offrent. Cette architecture fournit une vision plus claire du problème de l'adaptation et constitue ainsi une base solide pour des réflexions futures sur ce sujet.

Le troisième résultat est la conception d'un système d'édition interactif de présentations multimédias obtenues par transformation. Le succès du Web est en grande partie dû à la facilité de conception de pages HTML. Pour rendre accessible la conception de sites multimédias adaptables, il nous a semblé indispensable de fournir un outil d'édition interactif et convivial qui réduit la complexité intrinsèque d'un langage de transformation tel que XSLT.

Enfin, le dernier résultat majeur de cette thèse est la conception d'un moteur de transformation incrémental. L'utilisabilité du système d'édition que nous avons proposé, et de façon générale, le principe d'interactivité mis en œuvre, dépend fortement du temps de réponse du système. Il est donc indispensable de mettre à jour de façon optimale le résultat de la transformation en utilisant des techniques incrémentales.

3. Résultats pratiques

Dans cette thèse, la mise en œuvre pratique des résultats théoriques a représenté une part importante du travail accompli. La partie la plus importante du développement a été réalisée dans le prototype Madeus, dans lequel nous avons ajouté les fonctions d'adaptation et d'édition de documents adaptables.

Grâce à ce système, qui est aujourd'hui opérationnel, nous avons été en mesure d'appliquer les principes d'adaptation à des domaines très variés : l'aéronautique, les documents Docbook, les documents SlideShow et d'autres documents non décrits dans ce mémoire comme les albums de photos. Nous avons ainsi validé notre approche pour présenter des documents respectant des modèles. De plus, nous avons identifié plusieurs limites d'adaptation de notre système. Nous donnons quelques pistes pour les résoudre dans les perspectives de cette thèse.

Ce système nous a permis également d'exploiter de façon intensive le langage XSLT, qui est un langage très récent (la recommandation date du 16 Novembre 1999). Nous avons été en mesure d'évaluer ce langage selon plusieurs critères comme l'expressivité, la performance, son adéquation avec les différents besoins de transformation, la facilité d'écriture, etc. Il en ressort que XSLT est un langage très expressif, il permet de générer les présentations que nous souhaitions, au détriment de sa facilité d'utilisation. En effet, une des principales difficultés de XSLT est son caractère fonctionnel qui nécessite d'utiliser intensivement la récursivité.

La seconde partie des résultats pratiques se retrouve dans le processeur de transformation XSLT. Une des incertitudes que nous avions à son sujet portait sur sa capacité à être utilisé au sein d'un système d'édition. L'implantation de notre outil d'édition avec un processeur de transformation XSLT incrémental montre des résultats très prometteurs sur cet aspect, la modification d'un document XML à travers une vue de présentation s'effectue avec des temps de réponse acceptables. Certes, nous avons mené des expériences sur des documents de petite taille mais il faut noter que nous n'avons pas optimisé le traitement de formatage.

4. Perspectives

Cette thèse représente une première expérimentation pour la construction d'un système d'édition multimédia générique et adaptable. Elle fournit les bases à partir desquels il est

possible d'améliorer les résultats obtenus. En plus de ces améliorations, plusieurs perspectives à ces travaux sont envisageables. Nous avons organisé ces perspectives autour de quatre domaines de recherche abordés dans ce mémoire : la modélisation, la transformation, l'édition et l'adaptation.

4.1. Modélisation

Plusieurs fois dans ce chapitre nous avons souligné le manque d'expressivité des méta-modèles. En effet, actuellement ils se limitent à exprimer des contraintes de nature syntaxique. Plusieurs travaux sont en cours pour modéliser des contraintes de nature sémantique [RDFSchema 00, OMG 00, Unisys 01]. Les résultats obtenus jusqu'à présent dans ce domaine sont encore relativement incomplets. En particulier, ils ne prennent pas en compte les besoins de description de la sémantique de présentation ou de la linguistique.

4.2. Transformation

Plusieurs extensions et perspectives peuvent être envisagées au niveau de la vérification de la cohérence des transformations.

Vérification

- **Vérification statique de type (Static Type Checking).** Un des problèmes de XSLT est l'impossibilité de garantir de façon statique, c'est-à-dire avant l'exécution d'une transformation, que quelque soit le document source en entrée d'un processeur de transformation, le document produit respecte un modèle de document. En particulier, il n'y a aucune garantie que les feuilles de transformation générées par notre système d'édition produisent un document de présentation qui soit sémantiquement cohérent.

Transformation incrémentale

- **Extension à XSLT 2.0.** Une première perspective à cours terme est d'étendre la modélisation formelle du langage XSLT 1.0 décrite dans ce mémoire à l'ensemble des traits de ce langage, comme à la seconde version de ce langage : XSLT 2.0 [XSLT2.0 01]. Pour ce dernier, il serait intéressant d'étudier dans quelle mesure les fonctions incrémentales que nous avons présentées peuvent s'appliquer ;

- **Amélioration des performances.** Une seconde perspective est d'étudier comment améliorer encore les performances de la transformation incrémentale en tirant profit de la connaissance du modèle du document source. Ce modèle peut servir pour calculer des règles de réévaluation encore plus précises. De plus, il peut permettre de réduire le temps d'exécution d'une transformation et aussi de diminuer la taille mémoire requise par le document source en ne conservant que les fragments nécessaires à la transformation. Cela permettra également d'améliorer les performances d'un processeur de transformation classique (non incrémental) ;

- **Mise à jour de sites Web.** Nous avons vu qu'une des applications de la transformation était la mise à jour d'importants sites Web, pour lesquels le temps de construction peut devenir relativement long. Dans ce contexte, il parait naturel d'étudier si une mise à jour incrémentale est possible. En effet, par rapport à l'édition, la mise à jour de site s'opère à travers plusieurs sessions de modifications. Or, la transformation incrémentale repose

sur une structure intermédiaire dynamique. Faut-il alors chercher à restaurer cette structure à chaque demande de mise à jour du site ? Si oui, le temps de restauration n'est-il pas plus long que la mise à jour complète du site ? Ou alors, existe-t-il des situations pour lesquelles cette structure est inutile ?

- **Définition d'un profil « transformation incrémentale ».** La transformation incrémentale est coûteuse à cause de plusieurs facteurs relatifs à l'expressivité du langage. En particulier, dans XSLT il est possible de sélectionner des noeuds quel que soit leur emplacement dans l'arbre et quel que soit leur type. Est il intéressant de définir un profil restreint appelé « transformation incrémentale » permettant d'implanter des processeurs incrémentaux performants ? Si oui, est-il alors possible de passer automatiquement d'un profil regroupant l'ensemble des fonctionnalités de XSLT vers le profil incrémental.

4.3. Édition

Lors de la description de notre système d'édition, nous avons posé plusieurs restrictions. Dans cette section, nous donnons en perspectives des pistes qui pourraient permettre de lever ces restrictions. Nous avons organisé les perspectives selon qu'elles portent sur l'édition du document source ou sur l'édition des feuilles de transformation.

Au niveau du document source

- **Édition des nœuds source indirects.** Nous avons restreint l'édition du document source aux seuls nœuds de type direct, c'est-à-dire ceux qui produisent directement un résultat. L'édition des nœuds indirects, c'est-à-dire ceux qui contrôlent la transformation, s'effectue grâce à la génération par transformation de l'interface graphique. Est-il possible de caractériser de façon plus sémantique ces nœuds pour pouvoir déduire des actions communes à tous les modèles de documents source ?

- **Extension du modèle de description d'interfaces graphiques**. Le langage UIML permet de décrire des interfaces graphiques et communique avec l'application à travers un ensemble d'événements de type condition-action. Par contre, il ne permet pas d'accéder aux propriétés du contexte d'édition, comme la position du curseur textuel. L'interface graphique générée n'est alors pas aussi conviviale quelle pourrait l'être si une telle extension était apportée à UIML ;

- **Prise en compte du modèle source.** De nombreux systèmes actuels tirent profit de la connaissance du modèle du document source pour améliorer l'édition. Ils se limitent cependant à un niveau d'édition textuel, en offrant des services de complétion et de validation. L'étape suivante est d'étudier comment réutiliser ces principes mis en œuvre dans ces systèmes lorsque l'édition s'effectue après un processus de transformation et de formatage.

Au niveau des feuilles de transformation

- **Édition de l'interface graphique.** Nous avons défini plusieurs actions utilisateur pour éditer les feuilles de transformation dont le but est de produire une présentation multimédia. Cette approche peut facilement se généraliser à n'importe quel modèle de présentation, en particulier les modèles de description d'interfaces graphiques. Il devient

alors possible de concevoir facilement des éditeurs de documents spécialisés à un modèle source donné.

- **Amélioration de l'interface graphique.** Plusieurs extensions peuvent être apportées au niveau de l'interface graphique en proposant notamment une visualisation de la transformation. En effet, pour le moment, l'édition de la transformation est limitée en partie à cause du fait que la spécification de la transformation est totalement cachée à l'utilisateur. De ce fait, il n'est pas possible, entre autres, d'appliquer plusieurs fois la même règle. Une solution a été décrite dans [Villard 01d] en proposant une vue graphique semi-formatée de la transformation et une vue pour les règles de transformation.

Pour terminer, nous n'avons pas testé notre système d'édition auprès de « vrais utilisateurs ». C'est une tâche qui mériterait d'être réalisée.

4.4. Adaptation

Dans le chapitre 4, nous avons proposé une architecture pour réaliser un système adaptable. Plusieurs perspectives peuvent être envisagées sur ce thème de recherche.

- **Négociation**. Nous n'avons pas détaillé le fonctionnement du superviseur d'adaptation. De façon générale, le rôle du superviseur est de *négocier* une présentation qui correspond au mieux au contexte utilisateur. Ce travail constitue un thème de recherche à part entière et fait l'objet de la thèse mené par Tayeb Lemlouma au sein de l'équipe Opéra ;

- **Couplage fort des composants.** Il y a certains types d'adaptation qui ne sont pas couverts par le système adaptable que nous avons défini. Notamment, nous ne prenons pas en compte les préférences utilisateur qui portent à la fois sur le domaine et sur la présentation. Par exemple, si un auteur veut que la durée de la présentation SlideShow ne dure pas plus de 10 minutes, une solution d'adaptation est de ne présenter que les transparents les plus importants (selon des critères donnés). Cependant, la durée effective de la présentation d'un transparent ne peut être connue que lors du formatage, et donc le processus de transformation ne peut pas connaître le nombre de transparents à sélectionner. Ce couplage fort entre la transformation et le formatage peut se traduire par un besoin de spécification plus abstrait du formatage en intégrant des contraintes autres que temporelles et spatiales. Ce dernier aspect fait partie du sujet de thèse menée au sein de l'équipe Opéra par Frédéric Bes ;

- **Transformation pour l'adaptation.** Lors de la définition du système adaptable nous avons abordé la transformation surtout du point de vue de l'utilisateur. Certes, notre étude s'est restreinte au langage XSLT. La question est de savoir s'il est pertinent de rechercher d'autres langages de transformation plus appropriés pour l'adaptation de document.

- **Prise en compte des paramètres de l'utilisateur.** Dans ce mémoire, nous nous sommes focalisé sur l'adaptation de documents multimédias à différents terminaux ayant des capacités hétérogènes d'accès au réseau. Dans l'état de l'art, nous avons vu que l'adaptation ne se limite pas uniquement à ces aspects. Il serait intéressant d'étudier dans

quelle mesure le système adaptable que nous avons proposé permet de prendre en compte les paramètres propres à l'utilisateur, comme ses connaissances et ses capacités.

Références

1. Bibliographie

[Adiba 95] Michel Adiba, *"STORM: a Structural and Temporal Object-oRiented model for Multimedia databases"*, International Workshop on Multi-Media Database Management Systems (IW-MMDBMS), 1995.

[Agrawal 91] Hiralal Agrawal, Richard A. DeMillo et Eugene H. Spafford, *"An execution-Backtracking Approach to Debugging"*, IEEE Software, pp.21-26, Mai 1991.

[Akpotsui 93] Extase Akpotsui, *"Transformation de types dans les systèmes d'édition de documents structurés"*, Thèse en Informatique, INPG, Octobre 1993.

[Allen 83] J.F. Allen, *"Maintaining Knowledge about Temporal intervals"*, Communication of the ACM, Vol. 26, N° 11, pp. 832-843, Novembre 1983.

[Allen 90] R.B. Allen, *"User Models: Theory, Method, and Practice"*, International Journal of Man-Machine Studies, Vol. 32, pp. 551-543, 1990.

[Allison 87] Lloyd Allison, *"A Practical Introduction to Denotational Semantics"*, Cambridge University, 1987.

[André 89] Jacques André, Richard Furuta et Vincent Quint, *"Structured documents",* Cambridge University Press, Cambridge, 1989.

[Attali 01] Isabelle Attali, Carine Courbis, Pascal Degenne, Alexandre Fau, Didier Parigot et Claude Pasquier, *"SmartTools: a Generator of Interactive Environments Tools"*, In Reinhard Wilhelm, International Conference on Compiler Construction CC'01, Vol. 2027, Genève, Italie, Avril 2001.

[Anderson 01] Corin R. Anderson, Pedro Domingos et Daniel S. Weld, *"Personalizing Web Sites for Mobile Users"*, In Proceedings of the 10th Conference on the World Wide Web (WWW10), Mai 2001.

[Audebaud 00] Philippe Audebaud et Kristoffer Rose, *"Stylesheet Validation"*, Rapport de Recherche du Laboratoire de l'Informatique du Parallélisme (LIP), N° 2000-37, Novembre 2000.

[Ausburn 78] L. J. Ausburn et F. B. Ausburn, *"Cognitive Styles: Some Information and implications for Instructional Design"*, Educational Communication and Technology, Vol.26, pp. 337-354, 1978.

[Balbo 94] Sandrine Balbo, *"Évaluation ergonomique des interfaces utilisateur : un pas vers l'automatisation"*, Thèse en Informatique, Université Joseph Fourier, 5 Septembre 1994.

[Bastien 93] J.M. Christian Bastien et Dominique Scapin, *"Critères Ergonomiques pour l'Évaluation d'Interfaces Utilisateurs",* Rapport technique n°156, INRIA, Mai 1993.

[Badros 99] Greg Badros, Alan Borning, Kim Marriott et Peter J. Stuckey, *"Constraint cascading style sheets for the Web"*, In Proceedings of the 1999 ACM Conference on User Interface Software and Technology, Novembre 1999.

[Badros 01] Greg Badros, Jojada J. Tirtowidjojo, Kim Marriott, Bernd Meyer, Will Portnoy et Alan Borning, *"A constraint extension to scalable vector graphics"*, In Proceedings of the 10th Conference on the World Wide Web (WWW10), Mai 2001.

[Bickmore 99] Tim Bickmore, Andreas Girgensohn, et Joseph W. Sullivan, *"Web Page Filtering and Re-Authoring for Mobile Users"*, The Computer Journal, Vol. 42, N° 6, pp. 534-546, 1999.

[Björk 99] Staffan Björk, Lars Erik Holmquist, Johan Redström, Ivan Bretan, Rolf Danielsson, Jussi Karlgren et Kristofer Franzén, *"WEST: a Web browser for small terminals"*, In Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology, ACM, pp. 187-196, 1999.

[Boll 99] Susanne Boll, Wolfgang Klas et Utz Westermann, *"Multimedia Document Formats - Sealed Fate or Setting Out for New Shores?"*, In Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS 99), Florence, Italie, 7-11 Juin 1999.

[Boll 00] Susanne Boll et Wolfgang Klas, *"-ZYX – A Multimedia Document Model for Reuse and Adaptation of Multimedia Content",* Transaction on Knowledge and Data Engineering, DS-8 Special Issue, IEEE, 2000.

[Bonhomme 98] Stéphane Bonhomme, *"Transformations de documents structurés : une combinaison des approches déclaratives et automatiques",* Thèse en Informatique, Université Joseph Fourier, Décembre 1998.

[Borras 88] P. Borras, D. Clément, Th. Despeyroux, J. Incerpi, G. Kahn, B. Lang, et V. Pascual, *"Centaur: the system"*, In Proceedings of SIGSOFT'88, Third Annual Symposium on Software Development Environments (SDE3), Boston, USA, 1988.

[Boyle 94] Craig Boyle et Antonio O. Encarnacion, *"Metadoc: An Adaptive Hypertext Reading System"*, User Modeling and User-Adapted Interaction, Vol. 4, pp. 1-19, 1994.

[Briet 51] Susanne Briet, *"Qu'est-ce que la Documentation"*, Paris, Éditions Documentaires Industrielles et Techniques (ÉDIT), pp. 47, 1951.

[Buchanan 95] M. Cecilia Buchanan et Polle T. Zellweger, *"Automatically generating consistent schedules for multimedia documents"*, ACM-Multimedia Systems Journal, Vol. 1, N° 2, pp.55-67, 1995.

[Buckland 97] M.K. Buckland, *"What Is a Document?"*, Journal of the American Society of Information Science, Vol. 48, N° 9, pp. 804-809, 1997.

[Bulterman 95] Dick C.A. Bulterman et Lynda Hardman, *"Multimedia Authoring Tools: State of the Art and Research Challenges"*, Springer LNCS-1000, pp. 575-591, 1995.

[Buyukkokten 00] Orkut Buyukkokten, Hector Garcia Molina, Andreas Paepcke et Terry Winograd, *"Power Browser: Efficient Web Browsing for PDAs"*, In Proceedings of Human-Computer Interaction Conference (CHI), 2000.

[Carberry 88] S. Carberry, *"Modeling the User's Plans and Goals",* Comp. Linguistics, Vol. 14, N° 3, pp. 23-37, 1988.

[Calvary 01] Gaëlle Calvary, Joëlle Coutaz et David Thevenin, *"Supporting Context Changes for Plastic User Interfaces: A Process and a Mechanism"*, Joint Proceedings of HCI 2001 and IHM 2001, pp. 349-364, Lille, France, 2001.

[Carr 77] B. Carr et I. Goldstein, *"Overlays: A Theory of Modelling for Computer Aided Instruction",* Technical Report AI Memo 406, MIT, Cambridge, MA, 1977.

[Chabert 00] Sylvie Chabert-Ranwez, *"Composition Automatique de Documents Hypermédia Adaptatifs à partir d'Ontologies et de Modèles intentionnels de l'utilisateur"*, Thèse en Informatique, Université de Montpellier II, 21 Décembre 2000.

[Chen 00] Jinlin Chen, Yudong Yang et Hongjiang Zhang, *"An Adaptive Web Content Delivery System"*, International Conference on Adaptive Hypermedia and Adaptive Web-based Systems (AH2000), 28-30 Août 2000.

[Chen 01] Jinlin Chen, Baoyao Zhou, Jin Shi, Hongjiang Zhang et Qiufeng Wu, *"Function-based Object Model Towards Website Adaptation"*, In Proceedings of The Tenth International World Wide Web Conference, Hong Kong, 1-5 Mai 2001.

[Chin 89] David N. Chin, *"KNOME: Modeling What the User Knows in UC"*, User models in dialog systems, Springer Verlag, pp. 74-107, Berlin, 1989.

[Chin 93] David N. Chin, *"Acquiring User Models"*, Artificial Intelligence Review, Vol. 7, N° 3-4, pp. 185-197, 1993.

[Clark 01a] James Clark, *"TREX - Tree Regular Expressions for XML Language Specification"*, 2001. http://www.thaiopensource.com/trex/spec.html.

[Clark 01b] James Clark et Makoto Murata, *"RELAX NG Specification"*, Working Draft, 17 Septembre 2001.
http://www.thaiopensource.com/relaxng/spec.html.

[Coutaz 90] Joëlle Coutaz, *"Interface homme-ordinateur : conception et réalisation"*, Dunod, Paris, France, 1990.

[Dabbous 01] Walid Dabbous, *"Systèmes multimédias communicants"*, Réseaux et Télécoms, Edition Hermes, Juin 2001.

[Dalal 96] Mukesh Dalal, Steven Feiner, Kathleen McKeown, Shimei Pan, Michelle Zhou Tobias Höllerer, James Shaw, Yong Feng et Jeanne Fromer, *"Negotiation for Automated Generation of Temporal Multimedia Presentations"*, In ACM Multimedia 96, pp 55-64, 1996.

[Diaz 01] Victor Dias, Jérome Euzenat et Nabil Layaïda, *"Approche sémantique de l'adaptation de documents multimédia"*, Rapport de recherche, INRIA, Octobre 2001.

[Diehl 00] Stephan Diehl et Jörg Keller, *"VRML with constraints"*, In Proceedings of the Web3D-VRML, 2000.

[Dufresnen 97] Aude Dufresnen et Sylvie Tucotte, *"Cognitive style and its implications for navigation strategies"*, In Artificial Intelligence in Education: Knowledge and Media in Learning System, B. d. Boulay and R. Mizoguchi, 1997.

[Duluc 00a] Franck Duluc, *"Modélisation d'un fonds documentaire multimédia. Application à la documentation technique aéronautique"*, Thèse en Informatique, Institut de Recherche en Informatique de Toulouse - Université Paul Sabatier – Toulouse III, Décembre 2000.

[Duluc 00b] Franck Duluc, Cécile Roisin, Laurent Tardif et Lionel Villard, *"Un système multimédia complet pour la documentation technique aéronautique*", L'Objet, numéro thématique : Objets et multimédia - Éditions Hermès, Vol. 6, N° 2, pp. 223-253, 2000.

[Ekman 75] P. Ekman et W. Friesen, *"Unmasking the Face: A Guide to Recognizing Emotions from Facial Expresssions"*, Englewood Cliffs, NJ: Prentice Hall, 1975.

[Ellis 91] C. A. Ellis, S. J. Gibbs et G. L. Rein, *"Groupware, some issues and experiences",* Communications of the ACM, Vol. 34, N° 1, pp. 38-58, Janvier 1991.

[Erwig 99] M. Erwig, R.H. Güting, M. Schneider et M. Vazirgiannis, *"Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases"*, In GeoInformatica, Vol. 3, N° 3, pp. 269-296, 1999.

[Fink 98] Josef Fink, Alfred Kobsa et Andreas Nill, *"Adaptable and Adaptive Information Provision for All Users, Including Disabled and Elderly People"*, New Review of Hypermedia and Multimedia, Vol. 4, pp. 163-188, 1998.

[Fischer 01] Gerhard Fischer, "*User Modeling in HC",* International Journal of user modelling and user-adapted interaction, Vol. 11, pp. 65-86, 2001.

[Fox 96] Armando Fox, Steven D. Gribble, Eric A. Brewer et Elan Amir, *"Adapting to Network and Client Variability via On-Demand Dynamic Distillation"*, In Proceedings of the Seventh Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII), Cambridge, MA, Octobre 1996.

[Freire 01] Juliana Freire, Bharat Kumar et Daniel Lieuwen, *"WebViews: Accessing Personalized Web Content and Services"*, In Proceedings of the 10th Conference on the World Wide Web (WWW10), Mai 2001.

[François 97] Patricia François, *"Étude et conception d'une structure d'accueil de documents hypermédias structurés selon les normes SGML et HyTime. Application à la documentation aéronautique"*, Thèse en Informatique, Institut de Recherche en Informatique de Toulouse - Université Paul Sabatier – Toulouse III, 1997.

[Furuta 82] Richard Furuta, Jeffrey Scofield et Alan Shaw, *"Document Formatting Systems: Survey, Concepts and Issues"*, ACM Computing Surveys, Vol. 14, N° 3, pp. 417-472, Septembre 1982.

[Furuta 88a] Richard Furuta et P. David Stotts, *"Specifying Structured Document Transformations"*, Document Manipulation and Typography, In Proceedings of the internationnal Conference on Electronic Publishing, Cambridge University Press, Nice, 20-22 Avril 1988.

[Furuta 88b] Richard Furuta, Vincent Quint et Jacques André, *"Interactively Editing Structured Documents"*, Electronic Publishing -- Origination, Dissemination and Design, Vol. 1, N° 1, pp. 19-44, Avril 1988.

[Garlatti 99] Serge Garlatti, S. Iksal et P. Kervella, *"Adaptive On-Line Information System by means of a Task Model and Spatial Views"*, In Proceedings of the 2nd Workshop on Adaptive Systems and User Modeling on the WWW, pp. 59-66, 1999.

[Goldfarb 81] C. F. Goldfarb, *"A generalized approach to document markup"*, SIGPLAN Notices of the ACM, Juin 1981.

[Gutwin 99] Carl Gutwin et Saul Greenberg, *"A framework of awareness for small groups in shared-workspace groupware"*, Technical Report, Department of Computer Science, University of Saskatchewan, 1999.
http://www.cpcs.ucalgary.ca/papers/1999/99-AwarenessTheory/html/theory-tr99-1.html

[Guyard 00] Julien Guyard, *"Schéma Filmothèque"*, 2000.
http://www.inrialpes.fr/opera/film.xsd, 2000.

[Hauser 00] J. Hauser et K. Rothermel, *"Specification and Implementation of an Extensible Multimedia System"*, In Proceeding of Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS2000), Springer, 2000.

[Hosoya 00] Haruo Hosoya, Jérome Vouillon et Benjamin C. Pierce, *"Regular Expression Types for XML"*, In Proceedings of the International Conference on Functional Programming (ICFP), 2001.

[Hsu 94] J. Hsu, W. Johnston et J. McCarthy, *"Active outlining for HTML documents: An X-mosaic implementation"*, In 2nd Int. World Wide Web Conference, Chicago, 1994.

[Huang 00] Anita W. Huang et Neel Sundaresan, *"A Semantic Transcoding System to Adapt Web Services for Users with disabilities"*, ASSEST'00, pp. 156-163, Novembre 2000.

[Ivory 01] Melody Y. Ivory, Rashmi R. Sinha et Marti A. Hearst, *"Empirically Validated Web Page Design Metrics"*, In CHI 2001, ACM Conference on Human Factors in Computing Systems, CHI Letters, Vol. 3, N°1, 2001.

[Jelliffe01] Rick Jelliffe, *"The Schematron Assertion Language 1.5"*, Academia Sinica Computing Centre, 2001.
http://www.ascc.net/xml/resource/schematron/Schematron2000.html

[Jourdan 98] Muriel Jourdan, Nabil Layaïda, Cécile Roisin, Loay Sabry-Ismail et Laurent Tardif, *"Madeus, an Authoring Environment for Interactive Multimedia Documents"*, ACM Multimedia'98, pp. 267-272, ACM, Bristol (UK), Septembre 1998.

[Kaasinen 00] Eija Kaasinen, Matti Aaltonen, Juha Kolari, Suvi Melakoski et Timo Laakko, *"Two Approaches to Bringing Internet Services to WAP Devices",* Ninth International Word Wide Web Conference, Amsterdam, 15-19 Mai 2000.

[Kamba 96] T. Kamba, S.Elson, T. Harpold, T. Stamper et P. Sukaviriya, *"Using small screen space more efficiently"*, Human Factors in Computing Systems (CHI), Vancouver, Canada, pp. 383-390, 1996.

[Kantrowitz 93] Mark Kantrowitz, *"Bibliography of Research in Natural Language Generation"*, Rapport technique, School of Computer Science Carnegie Mellon University Pittsburgh, PA 15213, Novembre 1993.

[Kass 88] Robert Kass et Tom Finin, *"A general user modelling facility"*, In Proceedings on Human Factors in Computing Systems, pp. 145-150, Mai 1988.

[Kautz 87] H. Kautz, *"A Formal Theory Of Plan Recognition"*, Thèse en Informatique, University of Rochester, 1987.

[Kay 00] Michael Kay, *"XSLT Programmer's Reference"*, Wrox, ISBN 1861003129, 2000.

[Klarlund 00] Nils Klarlund, Anders Møller et Michael I. Schwartzbach, *"DSD: A Schema Language for XML"*, ACM SIGSOFT Workshop on Formal Methods in Sofware Practice, Portland, Août 2000.

[Kennel 96] Andrea Kennel, Louis Perrochon et Alireza Darvishi, *"WAB: World Wide Web Access for Blind And Visually Impaired Computer Users"*, New Technologies in the Education of the Visually Handicapped, ACM SIGCAPH Bulletin, Juin 1996.
http://www.inf.ethz.ch/department/IS/ea/blinds.

[Knuth 84] D.E. Knuth, *"The TEXbook"*, Addison-Wesley, Reading, Massachusetts, 1984.

[Kobsa 93] Alfred Kobsa, *"User Modeling: Recent Work, Propects and Hazards"*, Adaptive User Interfaces: Principles and Practices, North-Holland, Amsterdam, 1993.

[Kobsa 95] Alfred Kobsa et Wolfgang Pohl, *"The user modeling shell system BGP-MS"*, User Modeling and User Adapted Interaction, Vol. 4, N° 2, pp 59-106, 1995.

[Kobsa 01a] Alfred Kobsa, *"Generic User Modeling Systems"*, User Modeling and User-Adapted Interaction, Kluwer Academic Publishers, Vol. 11, pp. 49-63, 2001.

[Kobsa 01b] Alfred Kobsa, Jürgen Koenemann et Wolfgang Pohl, *"Personalized Hypermedia Presentation Techniques for Improving Online Customer Relationships"*, The Knowledge Engineering Review, Vol. 16, N° 2, pp.111-155, 2001.

[Kuikka 95] Eila Kuikka et Martti Penttonen, *"Transformation of structured documents"*, Electronic Publishing - Origination, Dissemination and Design, Vol. 8, N° 4, pp. 319-341, 1995.

[Küpper 99], D. Küpper et A. Kobsa, *"User-Tailored Plan Generation"*, J. Kay, ed. UM99 User Modeling: Proceedings of the Seventh International Conference, Springer Verlag, pp. 45-54, 1999.

[Laforge 01] Maximilien Laforge, *"Gestion prédictive et réactive de la qualité de service pour les documents multimédia temporisés"*, Mémoire Ingénieur CNAM, INRIA Rhône-Alpes, 2001.

[Layaïda 96] Nabil Layaïda et Loay Sabry-Ismail, *"Maintaining Temporal Consistency of Multimedia Documents using Constraint Networks"*, Multimedia Computing and Networking 1996, M. Freeman, P. Jardetzky, H. M. Vin, ed., pp. 124-135, SPIE 2667, Janvier 1996.

[Layaïda 97] Nabil Layaïda, *"Madeus : Système d'édition et de présentation de documents structurés multimédia"*, Thèse en Informatique, Université Joseph Fourier, Juin 1997.

[Layaïda 99] Nabil Layaïda, *"Adaptabilité : pistes d'étude pour la définition d'une infrastructure d'accès au contenu multimédia pour des machines hétérogènes"*, Rapport interne, INRIA Rhône-Alpes, 1999.

[Lee 00] Dongwon Lee, Wesley W. Chu, *Comparative Analysis of Six XML Schema Languages,* ACM SIGMOD Record, Vol. 29, N° 3, Septembre, 2000.

[Lemlouma 01] Tayeb Lemlouma et Nabil Layaïda, *"The Negotiation of Multimedia Content Services in Heterogeneous Environments"*, In Proceedings of The 8th International Conference on Multimedia Modeling (MMM 2001), Amsterdam, 5-7 Novembre 2001.

[Lindén 94] Greger Lindén, *"Incremental updates in structured documents"*, Thèse en Informatique, Université d'Helsinki, 5 Février 1994.

[Lisetti 00] Christine Lisetti et Diane J. Schiano, *"Automatic Facial Expression Interpretation: Where Human-Computer Interaction, Artificial Intelligence and Cognitive Science Intersect"*, Pragmatics and Cognition (Special Issue on Facial Information Processing: A Multidisciplinary Perspective), Vol. 8, N° 1, pp. 185-235, 2000.

[Liu 95] Yanhong A. Liu et Tim Teitelbaum, *"Systematic derivation of incremental programs",* Science of Computer Programming, Vol. 24, N° 1, pp. 1-30, Février 1995.

[Ma 00] Wei-Ying Ma, Ilja Bedner, Grace Chang, Allan Kuchinsky et HongJiang Zhang, *"a framework for adaptive content delivery in heterogeneous network environments"*, Multimedia Computing and Networking 2000 (MMCN2000), pp. 25-27, Janvier 2000.

[Mamrak 89] Sandra A Mamrak, Michael J Kaelbling, Charles K. Nicholas et Michael Share, *"Chameleon: A System for Solving the Data-Translation Problem"*, IEEE Transactions on Software Engineering, pp. 1090-1108, Septembre 1989.

[McHugh 97] Jason McHugh, Serge Abiteboul, Roy Goldman, Dallan Quass et Jennifer Widom, *"Lore: A Database Management System for Semistructured Data"*, SIGMOD Record, Vol. 26, N° 3, pp. 54-66, 1997.

[Mendes 01] Emilia Mendes, Nile Mosley et Steve Counsell*, "Web Metrics-Estimating Design and Authoring Effort"*, In IEEE Multimedia, Vol. 8, N° 1, pp. 50-57, 2001.

[Metso 01] Maija Metso, Antti Koivisto et Jaakko Sauvola, *"Content Model for Mobile Adaptation of Multimedia Information",* Special issue on 1999 IEEE Third Workshop on Multimedia Signal Processing, Journal of VLSI Signal Processing, Vol. 29, N° 1 / 2, , Kluwer Academic Publishers, p. 115-128, Août/Septembre 2001.

[Michard 00] Alain Michard, *"XML: langage et application"*, Eyrolles, ISBN 2-212-09206-7, 2000.

[Microsoft 01a] Microsoft, *"XDR Schema Developer's Guide"*, 2001.
http://msdn.microsoft.com/library/en-us/xmlsdk30/htm/xmconxmldevelopersguide.asp, 2001.

[Milo 00] Tova Milo, Dan Suciu et Victor Vianu, *"Typechecking for XML Transformers"*, In Proceedings on Principles of Database Systems (PODS), 2000.

[Mohan 99] R. Mohan, J. R. Smith et Chung-Sheng Li, *"Adapting Multimedia Content for Universal Access"*, IEEE Transactions on Multimedia, Vol. 1, N° 1, Mars 1999.

[Munson01] Ethan V. Munson, *"Proceedings of the ACM Symposium on Document Engineering"*, ACM Press, Novembre 2001.

[Murata 01] Makoto Murata, Dongwon Lee et Murali Mani, *"Taxonomy of XML Schema Languages using Formal Language Theory"*, Extreme Markup Languages, Montreal, Canada, Août 2001.

[Mynatt 94] Elizabeth D. Mynatt et Gerhard Weber, "*Nonvisual Presentation of Graphical User Interfaces: Contrasting Two Approaches*", In Proceedings of the 1994 ACM Conference on Human Factors in Computing Systems (CHI'94), Boston, MA, 24-28 Avril 1994.

[Navarro 01] Patrice Navarro, *"Edition de documents multimedias SMIL",* Mémoire d'Ingénieur CNAM, 5 Février 2001.

[Netzer 94] Robert H. B. Netzer et Mark H. Weaver, *"Optimal Tracing and Incremental Reexecution for debugging Long-Running Programs"*, SIGPLAN, 1994.

[Ng 96] Raymond T. Ng et Jinhai Yang, *"An analysis of buffer sharing and prefetching techniques for multimedia systems"*, Multimedia Systems, pp. 55-69, 1996.

[OC 91] B. C. O'Connor, *"Selecting Key Frames of Moving Image Documents: A Digital Environment for Analysis and Navigation*", Microcomputers for Information Management, Vol. 8, N° 2, pp. 119-133, 1991.

[Ortony 88] A. Ortony, G.L. Clore et A. Collins, *"The cognitive structure of emotions"*, Cambridge University Press, Cambridge, UK, 1988.

[Paek 98] Seungyup Paek et John R. Smith, "*Detecting Image Purpose in World-Wide Web Documents"*, In IS&T/SPIE Symposium on Electronic Imaging: Science and Technology - Document Recognition, Juin 1998.

[Pair 68] C. Pair et A. Quere, *"Définition et étude des bilangages réguliers"*, Information and Control, Vol. 13, pp. 565-593, 1968.

[Pearl 90] J. Pearl, *"Heuristique - Stratégies de recherche intelligente pour la résolution de problèmes par ordinateur"*, Cépaduès, 1990.

[Pérez 96] M.J. Pérez-Luque et T.D.C Little, *"A Temporal Reference Framework for Multimedia Synchronization"*, IEEE Journal on Selected Areas in Communications (Special issue: Synchronization issues in Multimedia Communication), Vol. 14, N° 1, pp. 36-51, Janvier 1996.

[Perkowitz 00] Mike Perkowitz, Oren Etzioni, *"Towards adaptive Web sites: conceptual framework and case study"*, Artificial Intelligence, Vol. 118, N° 1-2, 2000.

[Petrucci 00] Lori Stefano Petrucci, Eric Harth, Patrick Roth, André Assimacopoulos et Thierry Pun, *"WebSound: a generic Web sonification tool allowing HCI researchers to dynamically create new access modalities"*, In Proceedings of the Conference on Human Factors in Computing Systems (CHI 2000), 1-6 Avril 2000.

[Pietriga 01] Emmanuel Pietriga, Vincent Quint et Jean-Yves Vion-Dury, *"VXT: A Visual Approach to XML Transformations"*, In Proceedings of the ACM Symposium on Document Engineering (DocEng'01), pp. 1-10, 2001.

[PostScript 91] *"PostScript Language"*, Adobe Systems Incorporated, Addison-Wesley, Menlo Park, California, Novembre 1991.

[Pugh 87] William Pugh, "*Incremental computation via function caching"*, Thèse en Informatique, Department of Computer Science, Cornell University, Août 1987.

[Quint 87] Vincent Quint, *"Une approche de l'édition structurée des documents",* Thèse d'État, Université Scientifique, Technologique et Médicale de Grenoble, Mai 1987.

[Quint 97] Vincent Quint, *"The Languages of Thot"*, 1997.
http://www.inrialpes.fr/opera/Thot/Doc/languages.htm

[Rabiner 93] Lawrence Rabiner et Biing-Hwang Juang, *"Fundamentals of Speech Recognition"*, Prentice Hall PTR, ISBN 0-13-015157-2, 1993.

[Ram 99] Ashwin Ram, Richard Catrambone, Mark J. Guzdial, Colleen M. Kehoe, D. Scott McCrickard et John T. Stasko, *"PML: Representing Procedural Domains for Multimedia Presentations"*, *IEEE* Multimedia, Vol. 6, N° 2, pp 40-52, Avril-Juin, 1999.

[Ramalingam 93] G. Ramalingam and T. Reps, *"A categorized bibliography on incremental computation"*, In Conference Record of 20th Annual ACM Symposium on Principles of Programming Languages, pp. 502-510, ACM, New York, Janvier 1993.

[Reiter 00] Ehud Reiter et Robert Dale, *"Building Natural Language Generation Systems"*, Cambridge University Press, 2000.

[Rekik 01] Yassine Aziz Rekik, *"Modélisation et manipulation des documents structurés : une approche modulaire, flexible et évolutive"*, Thèse en Informatique, Ecole Polytechnique Fédérale de Lausanne, Lausanne, 2001.

[Rich 79] Elaine. Rich, *"User Modeling via Stereotypes"*, Cognitive Science, Vol. 3, pp. 329-354, 1979.

[Roisin 94] Cécile Roisin et Irène Vatton, *"Merging Logical and Physical Structures in Documents"*, Electronic Publishing -- Origination, Dissemination and Design, Special Issue, Proceedings of the Fifth International Conference on Electronic Publishing, Document Manipulation and Typography, *EP94,* Vol. 6, N° 4, pp. 327-337, Avril 1994.

[Roisin 99a] Cécile Roisin, *"Documents multimédia structurés",* Habilitation à diriger des recherches, spécialité informatique, Institut National Polytechnique, Septembre 1999.

[Roisin 99b] Cécile Roisin, Tien Tran_Thuong et Lionel Villard, *"Integration of structured video in a multimedia authoring system"*, In Proceedings of the Eurographics Multimedia'99 Workshop, Springer Computer Science, ed., pp. 133-142, Milan, Septembre 1999.

[Rossi 01] Gustavo Rossi, Daniel Schwabe et Robson Guimarães, *"Desiging Personalized Web Applications",* In Proceedings of the 10th Conference on the World Wide Web (WWW10), Mai 2001.

[Rumbaugh 97] James Rumbaugh, Ivar Jacobson et Grady Booch, "*Unified Modelling Language Reference Manual*", Addison-Wesley, ISBN 0 201 30998, 1997.

[Rutledge 99] Lloyd Rutledge, Lynda Hardman, Jacco van Ossenbruggen and Dick C.A Bulterman, *"Mix'n'Match: Exchangeable Modules of Hypermedia Style"*, In Proceedings of the 10th ACM Conference on Hypertext and Hypermedia: Returning to Our Diverse Roots, Darmstadt, Germany, pp 179-188, 21-25 Février 1999.

[Sabry 99] Loay Sabry-Ismaïl, *"Schéma d'exécution pour les documents multimédia distribués",* Thèse en Informatique, Université Joseph Fourier, Janvier 1999.

[Salminen 01] Airi Salminen et Frank Wm. Tompa, *"Requirements for XML Document Database Systems"*, In Proceedings of the ACM Symposium on Document Engineering (DocEng'01), pp. 85-94, Novembre 2001.

[Schamber 96] L. Schamber, *"What is a Document ? Rethinking the Concept in Uneasy Times "*, In J. Am. Soc. Information Sciences, Vol. 47, N° 9, pp. 669-671, 1996.

[Schilit 01] Bill N. Schilit, Jonathan Trevor, David M. Hilbert et Tzu Khiau Koh, *"m-Links: An Infrastructure for Very Small Internet Devices"*, In Proceedings of the 7th Annual International Conference on Mobile Computing and Networking 2001, ACM Press, pp. 122-131, 16-21 Juillet 2001.

[Shneiderman 97] Ben Shneiderman, *"Designing the User Interface: Strategies for Effective Human-Computer Interaction"*, Addison-Wesley, ISBN 0201694972, 1997.

[Smith 86] S.L. Smith et J.N. Mosier, *"A design evaluation checklist for user-system interface software"*, Report #MTR-9480 DS_TR_84-358, The MITRE Corporation, Bedford, MA, 1982.

[Smith 98] John R. Smith et Li Chung-Sheng, *"Decoding image semantics using composite region templates"*, IEEE Workshop on Content-based Access of Image and Video Libraries (CBAIVL-98), Juin 1998.

[Song 96] J. Song, Y. Doganata, M. Kim, et A. Tantawi, *"Modeling Timed User-Interactions in Multimedia Document"*, In Proceedings of the IEEE International Conference on Multimedia Computing Systems, USA, Novembre 1996.

[Summer 95] Kristen Summer, *"Toward a Taxonomy of Logical Document Structures"*, In Electronic Publishing and the Information Superhighway: Proceedings of the Dartmouth Institute for Advanced Graduate Studies (DAGS '95), pp. 124 - 133, Boston, Mai 1995.

[Sundaresh 91] R.S. Sundaresh et P. Hudak, *"Incremental Computation via Partial Evaluation"*, In Symposium on Principles of Programming Languages, pp. 1-13, 1991.

[Takahashi 75] Masako Takahashi, *"Generalization of regular sets and their application to a study of context-free languages"*, Information and Control, Vol. 27, pp. 1-36, 1975.

[Tardif 00] Laurent Tardif, *"Kaomi: réalisation d'une boîte à outils pour la construction d'environnements d'édition de documents multimédia"*, Thèse en Informatique, INPG – Grenoble, Décembre 2000.

[TEI], *"The TEI Guidelines"*, Text Encoding Initiative, http://www.tei-c.org.

[Theodoridis 98] Y. Theodoridis, T. Sellis, A. Papadopoulos et Y. Manolopoulos, *"Specifications for Efficient Indexing in Spatiotemporal Databases"*, In Proceedings of 10th International Conference on Scientific and Statistical Database Management (SSDBM 98), Capri, Juillet 1998.

[Thevenin 99] David Thevenin et Joëlle Coutaz, *"Plasticity of User Interfaces: Framework and Research Agenda"*, In proceedings of INTERACT'99, pp. 110-117, Septembre 1999.

[TTran 01] Tien Tran_Thuong et Cécile Roisin, *"Structured Media for Multimedia Document Authoring"*, International Workshop on Web Document Analysis, 2001.

[Tozawa 01] Akihiko Tozawa, *"Towards Static Type Checking for XSLT "*, In Proceedings of the ACM Symposium on Document Engineering (DocEng'01), pp. 18-27, Novembre 2001.

[UAProf 99] *"User Agent Profile Specification"*, Wireless Application Protocol Forum, 10 Novembre 1999.
http://www1.wapforum.org/tech/terms.asp?doc=WAP-174-UAPROF-19991110-a.pdf.

[UIML 00] UIML.org, *"UIML v2.0 Draft Specification"*, 2000.
http://www.uiml.org/specs/uiml2/DraftSpec.htm

[Unisys 01] Unisys, *"The Unisys Universal Repository"*, 2001.
http://www.unisys.com/marketplace/urep/

[Vetro 01] Anthony Vetro, Huifang Sun et Yao Wang, *"Object-based transcoding for adaptable video content delivery"*, IEEE Transactions on Circuit and Systems for Video Technology, Vol. 11, N° 3, pp. 387-401, Mars 2001.

[Villard 00a] Lionel Villard, *"Spécification d'un modèle de document multimédia"*, Rapport d'avancement contractuel, INRIA, 15 Mars 2000.

[Villard 00b] Lionel Villard, Cécile Roisin et Nabil Layaïda, *"An XML-based multimedia document processing model for content adaptation"*, In Proceedings of the Eighth International Conference on Digital Documents and Electronic Publishing, 2000.

[Villard 01a] Lionel Villard, *"SMIL2.0 vers XHTML+SMIL : la feuille de transformation XSLT"*, 2001.
http://www.inrialpes.fr/opera/people/Lionel.Villard/SMIL/SMIL2toxhtmlplussmil.html

[Villard 01b] Lionel Villard, *"SMIL2.0 vers SMIL Basic : la feuille de transformation XSLT"*, 2001.
http://www.inrialpes.fr/opera/people/Lionel.Villard/SMIL/SMIL2tosmilbasic.html

[Villard 01c] Lionel Villard, *"Spécification d'un modèle de document aéronautique multimédia"*, Rapport d'avancement contractuel, INRIA, 30 Juin 2001.

[Villard 01d] Lionel Villard, *"Authoring Transformations by Direct Manipulation for Adaptable Multimedia Presentations"*, In Proceedings of the ACM Symposium on Document Engineering (DocEng'01), pp. 125-134, Atlanta, Georgia, 9-10 Novembre 2001.

[Villard 02] Lionel Villard et Nabil Layaïda, *"An Incremental XSLT Transformation Processor for XML Document Manipulation"*, In proceedings of the Eleventh International World Wide Web Conference (WWW2002), Hawaii, 7-11 Mai 2002.

[Waern 96] Annika Wærn, *"Recognising Human Plans: Issues for Plan Recognition in Human - Computer Interaction",* Thèse en Informatique, Akademitryck, Edsbruck, Norvège, 1996.

[Wadler 00] Philip Wadler, *"A formal semantics of patterns in XSLT"*, Markup Languages: Theory and Practice, Vol. 2, N° 2, pp. 183-202, 2000.

[Watanabe 99] Toyohide Watanabe, *"Document Analysis and Recognition"*, IEICE Trans. Inf. & Syst., Surveys on Image Processing Technologies, Vol. E82-D, N° 3, pp. 521-522, 1999.

[WebObject 01] *"Web Objects 5 Developer Documentation"*, Apple Computer, Inc, 2001.
http://developer.apple.com/techpubs/webobjects/.

[Xiong 00] Zixiang Xiong et Kannan Ramchandran, *"Wavelet image compression"*, Handbook of Image and Video Processing, A. Bovik, ed., Academic Press, 2000.

[XML-list] XML-dev mailing list, http://lists.xml.org/archives/xml-dev/.

[XUL 01] *"Mozilla XUL and Script Reference"*, 2001.
http://www.xulplanet.com/tutorials/xultu/elemref/

[Yeadon 96] Nicholas Yeadon, Francisco García, David Hutchison et Doug Shepherd, *"Filters: QoS Support Mechanisms for Multipeer Communications"*, In IEEE Journal on Selected Areas in Computing (JSAC) special issue on Distributed Multimedia Systems and Technology, Vol. 14, N° 7, pp 1245-1262, Septembre 1996.

[Zhang 93] L. Zhang, S. Deering, D. Estrin, S. Shenker et D. Zappala, *"RSVP: A New Resource ReSerVation Protocol"*, IEEE Network, September 1993.

[Ziegler 88] J. Ziegler, K. P. Fhnrich, *"Direct Manipulation"*, in M. Helander (Eds.), Handbook of Human-Computer Interaction, North-Holland: Elsevier Science Publishers, pp. 123-133, 1988.

2. Systèmes

[Amaya] *"Amaya W3C's Editor/Browser"*, Irène Vatton.
http://www.w3.org/Amaya/

[Balise] *"Balise Documentation – Release 4"*, AIS.
http://balise.xoasis.com/doc/doc.htm

[Cocoon] *"Apache Cocoon"*, The Apache Software Foundation.
http://xml.apache.org/cocoon/index.html

[DoCoMo01], *"SMILEditor Ver. 2"*, DoCoMo Systems.
http://www.docomo-sys.co.jp/prod/soft/smil2.html

[Dreamweaver 4] *"Dreamweaver 4"*, Macromedia.
http://www.macromedia.com/software/dreamweaver

[Director 8.5] *"Director 8.5 Shockwave Studio"*, Macromedia.
http://www.macromedia.com/fr/software/director/

[Echarpe] *"L'écharpe communicante"*, France Télécom.
http://www.electronicshadow.com/mediacol/

[EditMLPro, *"EditML Pro"*, NetBryx Technologies.
http://www.editml.com/

[GRiNS], *"GRiNS Editor"*, Oratrix.
http://www.oratrix.com/

[KOffice] *"KOffice"*, the KOffice Web team.
http://www.koffice.org

[LimSee] *"LimSee : Un éditeur temporel pour les documents au format SMIL"*, Projet Opéra, INRIA.
http://www.inrialpes.fr/opera/LimSee.html

[Merlot] *"Merlot"*, ChannelPoint.
http://www.merlotxml.org

[Microsoft 01b] Microsoft, *"XDR-XSD Converter"*, 2001.

http://msdn.microsoft.com/downloads/code/sample.asp?url=/msdn-files/027/001/539/msdncompositedoc.xml

[Morphon], *"Morphon XML-Editor 2.0"*, Morphon Technologies.
http://www.morphon.com

[Mozilla] *"Mozilla"*, The Mozilla Organization.
http://www.mozilla.org

[Omnimark5] *"Guide to OmniMark 5"*, Omnimark, 2001
http://www.omnimark.com

[PatML]. *"PatML"*, IBM.
http://www.alphaworks.ibm.com/tech/patml

[Screenfridge] *"Screenfridge"*, Electrolux.
http://www.electrolux.com/screenfridge/

[SAX] *"SAX"*, http://www.saxproject.org/

[StylusStudio] *"Stylus Studio"*, Excelon.
http://www.stylusstudio.com

[Thot] *"Thot structured document editor"*, Projet Opéra, INRIA Rhône-Alpes.
http://www.inrialpes.fr/opera/Thot/

[VisualXSLT] *"Visual XSLT"*, ActiveState.
http://aspn.activestate.com/ASPN/ Downloads/VisualXSLT

[WebDraw], *"Jasc WebDraw"*, Jasc Software.
http://www.jasc.com/products/webdraw/

[Xalan] *"Xalan-Java"*, The Apache Software Foundation.
http://xml.apache.org/xalan-j/index.html

[XMetal2.0], *"XMetal 2.0"*, SoftQuad.
http://www.xmetal.com

[XMLPro V2], *"XML Pro v2"*, Vervet Logic.
http://www.vervet.com

[<XML>Script] *"<XML>Script"*, Decision Soft.
http://www.xmlscript.org/index.html

[XMLSpy] *"XML Spy 4.1"*, Altova.
http://www.xmlspy.com

[<xsl>Composer] *"<xsl>Composer"*, Whitehill.
http://www.whitehill.com/Products/xslcomposer/

[XSLDebug] Chris Stefano, *"XSLDebugger, A visual XSLT debugger"*.
http://www.vbxml.com/xsldebugger/

3. Normes et recommandations

[ATA2100 00] *"ATA spécification 2100"*, Air Transport Association of America (ATA), 2000.

[CC/PP 01] *"Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies"*, W3C Working Draft, 15 Mars 2001.
http://www.w3.org/TR/CCPP-struct-vocab/

[CSS1 96] *"Cascading Style Sheets, level 1"*, W3C Recommandation, 17 Décembre 1996.
http://www.w3.org/ TR/REC-CSS1

[CSS2 01] *"Cascading Style Sheets, Level 2"*, W3C Recommandation, 12 mai 1998.
http://www.w3.org/ TR/REC-CSS2/

[DOM 00] *"Document Object Model (DOM) Level 2 Views Specification",* W3C Recommandation, 13 Novembre 2000.
http://www.w3.org/TR/DOM-Level-2-Views/

[FIPA 01] *"FIPA Device Ontology Specification"*, Foundation for Intelligent Physical Agents, Genève, Suisse, 09 Avril 2001.
http://www.fipa.org/specs/fipa00091/PC00091A.html

[Webdav 99] *"HTTP Extensions for Distributed Authoring – WEBDAV"*, Y. Goland, E. Whitehead, A. Faizi, et D. Jensen, Microsoft, U.C. Irvine, Netscape, Novell, RFC 2518, Février 1999.

[HTML 99] *"HTML 4.01 Specification"*, W3C Recommandation, 24 Décembre 1999.
http://www.w3.org/TR/html4/

[ISO 86] ISO, *Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML)*, (ISO 8879 :1986), International Organization for Standardisation, Genève, 1986.

[ISO 98] ISO/IEC/JTC1/SC29/WG11/IS-14496-2, *"Coding of audio vidual objects: visual"*, Mars 1998.

[ISO 00] ISO, *"Universal Multiple-Octet Coded Character Set (UCS) - - Part 1: Architecture and Basic Multilingual Plane (BMP) - IEC 10646"*, Seconde édition, Mars 2001.

[ISO 01] ISO/IEC JTC1/SC29/WG11, *"MPEG-7 Requirements Document, Coding of Moving Pictures and Audio"*, Mars 2001

[ITU 98] ITU-T-Study-Group-16, *"Draft Text of Recommandation H.263 Version 2 (H.263+) for decision"*, Octobre 1998.

[MQ 01] *"Media Queries"*, Håkon Wium Lie et Tantek Çelik, W3C Working Draft, 17 Mai 2001.
http://www.w3.org/TR/css3-mediaqueries

[Murata 01] Murata Makoto, *"RELAX (Regular Language description for XML)",* INSTAC (Information Technology Research and Standardization Center), 2001.
http://www.xml.gr.jp/relax/

[Namespace 99], *"Namespaces in XML"*, T. Bray, D. Hollander, A. Layman, W3C Recommandation, 14 January 1999
http://www.w3.org/TR/REC-xml-names

[Oasis 01] Oasis, *"Docbook"*, 2001.
http://www.docbook.org

[OMG 00] OMG, *"XML Metadata Interchange (XMI) version 1.1"*, Object Management Group, 2000.
http://www.omg.org/technology/documents/formal/xmi.htm

[P3P 01] *"The Platform for Privacy Preferences 1.0 (P3P1.0) Specification"*, W3C Working Draft, http://www.w3.org/TR/P3P/, 28 Septembre 2001.

[RDF 99] *"Resource Description Framework (RDF) Model and Syntax Specification"*, W3C Recommandation, 22 Février 1999.
http://www.w3.org/TR/REC-rdf-syntax/.

[RDFSchema 00] *"Resource Description Framework (RDF) Schema Specification 1.0"*, W3C Candidate Recommandation, 27 March 2000.
http://www.w3.org/TR/rdf-schema/

[RFC2506 01] *"Media Feature Tag"*, RFC2506, 23 Août, 2001.
http://www.iana.org/assignments/media-feature-tags

[Ruby 01] *"Ruby Annotation"*, W3C Recommandation, 31 Mai 2001.
http://www.w3.org/TR/2001/REC-ruby.

[Schema 01] *"XML Schema Part 0: Primer"*, David C. Fallside, W3C Recommandation, 2 Mai 2001.
http://www.w3.org/TR/xmlschema-0/

[SMIL 98] *"Synchronized Multimedia Integration Language (SMIL 1.0) Specification"*, W3C Recommandation, 15 Juin 1998.
http://www.w3.org/TR/REC-smil

[SMIL 01] *"Synchronized Multimedia Integration Language (SMIL 2.0) Specification"*, W3C Recommandation, 05 Août 2001.
http://www.w3.org/TR/smil20/

[SOAP 01] *"SOAP Version 1.2"*, W3C Working Draft, 9 Juillet 2001.
http://www.w3.org/TR/soap12/

[SOX 99] *"Schema for Object-Oriented XML 2.0"*, W3C Note 30, Juillet 1999.
http://www.w3.org/TR/ NOTE-SOX/

[SVG 01] *"Scalable Vector Graphics (SVG) 1.0 Specification"*, W3C Recommandation, 04 Septembre 2001.
http://www.w3.org/TR/SVG/

[VoiceXML 01] *"Voice Extensible Markup Language (VoiceXML) Version 2.0"*, W3C Working draft, 01 Juillet 2001.
http://www.w3.org/TR/voicexml/

[WAF 99] Wireless Application Forum, *"Wireless Application Environment Specification"*, 1999.
http://www.wapforum.org/what/technical/SPEC-WAESpec-19990524.pdf

[WebAccess 99] *"Web Content Accessibility Guidelines 1.0"*, W3C Recommandation, 5 Mai 1999.
http://www.w3.org/TR/WCAG10/

[XForms 01] *"XForms 1.0"*, W3C Working Draft, 07 Décembre 2001.
http://www.w3.org/TR/xforms/

[XHTML 00] *"XHTML 1.0: The Extensible HyperText Markup Language"*, W3C Recommandation, 26 Janvier 2000.
http://www.w3.org/TR/xhtml1/

[XHTML+SMIL 01] *"XHTML+SMIL Profile"*, Debbie Newman, Patrick Schmitz et Aaron Patterson, W3C Working Draft, 07 Août 2001.
http://www.w3.org/TR/XHTMLplusSMIL/

[XLink 01] *"XML Linking Language (XLink) Version 1.0"*, Steve DeRose, Eve Maler, David Orchard, W3C Recommandation, 27 Juin 2001.
http://www.w3.org/TR/xlink/

[XML 01] *"eXtensible Markup Language (XML) 1.0 (Second Edition)"*, Tim Bray, Jean Paoli and Co, W3C Recommandation, 6 Octobre 2001.
http://www.w3.org/TR/2000/REC-xml-20001006

[XPath 99] *"XML Path Language (XPath)"*, James Clark and Steve DeRose, Recommandation W3C, 16 November 1999.
http://www.w3.org/TR/xpath

[XSL 01] *"Extensible Stylesheet Language (XSL) Version 1.0"*, W3C Recommandation, 15 Octobre 2001.
http://www.w3.org/TR/xsl

[XSLT 99] *"XSL Transformations (XSLT)"*, James Clark, W3C Recommandation, 16 Novembre 1999.
http://www.w3.org/TR/xslt

[XSLT2.0 01] *"XSL Transformations (XSLT) Version 2.0"*, Michael Kay, W3C Working Draft, 20 Décembre 2001.
http://www.w3.org/TR/xslt20/

[SchemaDatatype 01] *"XML Schema Part 2: Datatypes"*, Paul V. Biron et Ashok Malhotra, W3C Recommandation, 02 Mai 2001.
http://www.w3.org/TR/xmlschema-2/

Annexe A

Feuille de transformation de référence pour la transformation incrémentale

```
1.  <?xml version="1.0" encoding="ISO-8859-1"?>
2.
3.  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4.      <xsl:output method="html"/>
5.
6.      <xsl:param name="toc.depth">2</xsl:param>
7.
8.      <xsl:template match="article">
9.          <html>
10.             <body>
11.                 <p align="center">
12.                     <xsl:value-of select="artheader/authorgroup/author/firstname"/>
13.                     <xsl:text> </xsl:text>
14.                     <xsl:value-of select="artheader/authorgroup/author/lastname"/>
15.                     presents
16.                 </p>
17.                 <h1 align="center"><xsl:value-of select="title"/></h1>
18.                 <xsl:apply-templates select="section">
19.                     <xsl:with-param name="indent">0</xsl:with-param>
20.                 </xsl:apply-templates>
21.                 <hr/>
22.                 <table border="1" width="100%">
23.                     <tr>
24.                         <td><xsl:apply-templates select="artheader/authorgroup"/></td>
25.                         <td>Nb uppest sections : <xsl:value-of select="count(section)"/></td>
26.                         <td>Last modification : <xsl:value-of select="artheader/date"/></td>
27.                     </tr>
28.                 </table>
29.             </body>
30.         </html>
31.     </xsl:template>
32.
33.     <xsl:template match="section">
34.         <xsl:param name="indent">0</xsl:param>
35.
36.         <xsl:variable name="heading">
37.             <xsl:choose>
38.                 <xsl:when test="count(ancestor::section) = 0">h2</xsl:when>
39.                 <xsl:when test="count(ancestor::section) = 1">h3</xsl:when>
40.                 <xsl:when test="count(ancestor::section) = 2">h4</xsl:when>
41.                 <xsl:otherwise>p</xsl:otherwise>
42.             </xsl:choose>
43.         </xsl:variable>
44.
```

```
45.       <xsl:element name="{$heading}">
46.          <xsl:attribute name="align">left</xsl:attribute>
47.          <xsl:attribute name="style">padding-left=
48.             <xsl:value-of select="$indent"/>px
49.          </xsl:attribute>
50.
51.          <xsl:number value="position()" format="1."/>
52.             <xsl:text> </xsl:text>
53.             <xsl:value-of select="title"/>
54.       </xsl:element>
55.
56.       <xsl:if test="count(ancestor::section) &lt; $toc.depth - 1">
57.          <xsl:apply-templates select="section">
58.             <xsl:with-param name="number-format">a.</xsl:with-param>
59.             <xsl:with-param name="indent" select="$indent + 100"/>
60.          </xsl:apply-templates>
61.       </xsl:if>
62.    </xsl:template>
63.
64.    <xsl:template match="authorgroup">
65.       <xsl:for-each select="author">
66.          <xsl:value-of select="firstname"/>
67.          <xsl:text> </xsl:text>
68.          <xsl:value-of select="lastname"/>
69.          <xsl:choose>
70.             <xsl:when test="position()=last() - 1"><xsl:text> and </xsl:text></xsl:when>
71.             <xsl:when test="position() &lt; last() - 1"><xsl:text>, </xsl:text></xsl:when>
72.             <xsl:otherwise/>
73.          </xsl:choose>
74.       </xsl:for-each>
75.    </xsl:template>
76.
77.    </xsl:stylesheet>
```