Une introduction à XML/XSLT

Bernd Amann et Philippe Rigaux
Cnam

13 mars 2009

Table des matières

Ce chapitre a pour ambition de proposer au lecteur une présentation intuitive des principales caractéristiques de XML, ainsi qu'un tour d'horizon de ses apports en terme d'intégration, d'échange et de

publication de données. Nous évitons délibérément, à ce stade, toute discussion technique détaillée, l'objectif étant plutôt de dresser un panorama des contextes d'utilisation de XML et de son association avec XSLT, en vue de convaincre le lecteur de l'intérêt de recourir à ces langages.

Il existe probablement plusieurs manières de lire ce chapitre. Une première lecture, rapide et laissant de côté les détails des quelques programmes proposés, suffira à comprendre intuitivement le rôle de XSLT et l'importance de ce langage dans les applications XML. Une seconde lecture permettra de saisir les principes de base de l'association XML/XSLT : structure arborescente des documents, traitement du document par des règles XSLT, et production du résultat par assemblage des fragments produits par les règles. Enfin nous avons placé dans ce chapitre quelques éléments de discussion sur des techniques avancées (notamment l'intégration avec les bases de données) qui fournissent une perspective générale sur les architectures incluant des documents XML et leur transformation par des programmes XSLT. Il ne faut donc pas hésiter à laisser de côté, au moins dans un premier temps, les aspects qui peuvent paraître obscurs ou rébarbatifs. La syntaxe des langages basés sur XML peut sembler déroutante à première vue, et la raison d'être de cette introduction est justement de proposer une initiation progressive qui permette de découvrir le rôle et l'intérêt de ces langages.

Nous souhaitons que ce premier chapitre permette, sans investir dans une étude technique approfondie, de dégager clairement la place de XML au sein des nombreux outils, langages et techniques qui constituent un système d'information orienté vers la publication ou les échanges de données sur le Web. Pour tous les aspects qui ne sont, techniquement parlant, qu'esquissés dans cette introduction, nous indiquons finalement le chapitre ou la partie du livre où le lecteur trouvera un développement complet.

Nous prenons comme fil conducteur, dans ce chapitre, une application simple qui nous permettra de décliner des exemples d'utilisation de XML. Cette application consiste à fournir, sous les formes les plus variées, les informations relatives aux films à l'affiche en France, comprenant un descriptif de chaque film, et les cinémas, salles et séances où ces films sont projetés. Nous supposons de plus que ces informations sont disponibles en différents points du réseau Internet. Nos exemples montreront comment XML permet d'échanger ces informations, de les intégrer, et enfin de les publier sur les supports les plus divers.

1 L'application et ses besoins

Décrivons tout d'abord l'application (simplifiée). L'objectif général consiste, pour un cinéma, à diffuser le plus largement possible l'information relative à ses salles, avec les films qui y sont diffusés et les horaires des séances. Nous prendrons comme exemple principal le cas du cinéma *L'Épée de bois* qui propose deux films :

– Le film *Alien*, de Ridley Scott, projeté dans la salle 1 avec trois séances dans la journée ;
– le film *Vertigo*, d'Alfred Hitchcock, projeté dans la salle 2 avec une seule séance à 22 heures.

L'Épée de bois souhaite bien entendu rendre ces informations disponibles sur son site web. Mais il envisage également le cas de cinéphiles munis d'un téléphone mobile, susceptibles de consulter les séances *via* une application WAP. Enfin le programme des salles doit être affiché à l'entrée du cinéma, distribué dans l'environnement proche (cafés, librairies) sous forme de tracts, et transmis à un magazine d'informations sur les spectacles.

Pour tous ces modes de diffusion, la solution traditionnelle, basée sur des outils adaptés à chaque cas, implique de resaisir l'information, avec des risques d'erreur multipliés et une perte de temps inutile. Par exemple la plaquette serait stockée au format propriétaire d'un traitement de mise en page, chaque site web placerait le contenu dans des fichiers HTML, le magazine conserverait ce même contenu dans une base de données, et ainsi de suite.

Supposons de plus qu'un site, *www.sallesenligne.com*, propose de référencer toutes les séances de tous les cinémas en France, et offre aux internautes un service de recherche et d'indexation. Bien entendu cela suppose que chaque cinéma lui fournisse, dans un format donné, les informations sur ces propres séances, ce qui implique pour *l'Épée de bois* un travail supplémentaire de saisie et mise à jour.

Enfin on supposera que des informations sur des films (acteurs, résumé) sont disponibles dans une base de données interrogeable sur le web. Il serait souhaitable que cette information puisse également être intégrée au programme pour le rendre encore plus attrayant.

En résumé, la problématique est double : d'une part il faut être en mesure de fournir une même information – le programme de cinéma – sous les formes les plus variées, d'autre part il faut « récupérer » tout ce qui peut enrichir cette information, et intégrer le tout dans un format cohérent. Nous allons explorer dans la suite de ce chapitre comment XML/XSLT répond à ce double besoin.

2 XML, format universel

XML constitue un moyen de rendre un même contenu accessible à plusieurs applications. Considérons le cas des informations propres au cinéma, à savoir son nom, son adresse et la station de métro la plus proche :

```
L'Epée de bois, 100 rue Mouffetard, métro Censier-Daubenton
```

Ces quelques informations constituent un *contenu* susceptible d'apparaître sur de nombreux supports différents : des affiches de film, un magazine des spectacles à Paris, de très nombreux sites web, des plaquettes commerciales, un téléphone portable, etc. Dans un contexte cloisonné où ces différents supports sont produits indépendamment les uns des autres, ce contenu est habituellement dupliqué autant de fois que nécessaire, et associé à un format propre à chaque support.

Dans la mesure où les applications gérant ce contenu ne communiquent pas, cette duplication et cette hétérogénéité des formats, adaptés à chaque type d'exploitation de ce contenu, sont légitimes. Si, en revanche, on se place dans un environnement connecté au réseau et favorisant les échanges d'information, duplication et hétérogénéité deviennent beaucoup moins justifiables. La duplication induit un coût, de transformation ou de stockage, et l'hétérogénéité peut rendre inaccessible ou inexploitable une information pourtant présente.

2.1 Qu'est-ce que XML ?

XML est donc d'abord destiné à représenter des contenus indépendamment de toute application. Il s'appuie pour cela sur la combinaison de plusieurs principes simples et généraux pour la représentation et l'échange d'information. Voici une représentation XML de notre cinéma :

```
<?xml version="1.0" encoding="ISO-8859-1"?><CINEMA><NOM>Epée de Bois
</NOM><ADRESSE>100, rue Mouffetard</ADRESSE><METRO>Censier-Daubenton
</METRO></CINEMA>
```

Une information codée en XML est donc simplement représentée sous forme d'une *chaîne de caractères*. Cette chaîne débute par une *déclaration XML* :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Cette ligne indique que la chaîne contient des informations codées avec la version 1.0 de XML, et que le jeu de caractères utilisé est conforme à la norme ISO-8859 définie par l'Organisation Internationale de Standardisation (ISO) pour les langues latines. Cette norme est adaptée à l'usage du français puisqu'elle permet les lettres accentuées commme le `'é'` dans le mot `Epée`.

2.2 Structuration avec XML

La représentation sous forme de chaîne de caractères n'exclut pas la structuration de l'information. Par exemple, la chaîne de caractères *"L'adresse du cinéma Épée de Bois est 100, rue Mouffetard et se trouve près de la station de métro Censier-Daubenton"* contient la même information que la chaîne XML précédente, mais est difficilement exploitable par un outil informatique car la structure de la phrase est cachée. Cette structure est marquée, en XML, par des *balises*[1] encadrées par les symboles < et >. Les balises <`CINEMA`>, <`NOM`>, <`/NOM`>, <`ADRESSE`>, <`/ADRESSE`>, <`METRO`>, <`/METRO`> et <`/CINEMA`> décomposent ainsi le contenu en trois parties textuelles :

[1] XML est un *langage de balisage structurel*.

– `Epée de Bois` ;
– `100, rue Mouffetard` ;
– `Censier-Daubenton`.

On peut constater que les balises forment des parenthèses autour de fragments de la chaîne XML. On trouve dans notre exemple les « parenthèses » <CINEMA>...</CINEMA>, <NOM>...</NOM>, <ADRESSE>...</ADRESSE> <METRO>...</METRO>. Dans ces paires de balises, la première est appelé *balise ouvrante* et la deuxième la *balise fermante*. Voici la terminologie établie pour désigner les constituants d'une chaîne XML :

– une paire de balises ouvrante et fermante et le fragment qu'elles entourent constituent un *élément XML* ;
– le nom de la balise est le *type de l'élément* ;
– le *contenu d'un élément XML* est obtenu en enlevant les balises qui l'entourent.

La structuration avec XML permet d'une part de *désigner* certaines parties du contenu avec des noms d'éléments, et d'autre part de *structurer* ce contenu en définissant une hiérarchie entre les éléments. Une des principales règles de structuration est en effet que le parenthésage défini par les balises doit être imbriqué : si une balise <B> est ouverte entre deux balises <A> et </A> définissant un élément, elle doit également être fermée par </B> entre ces deux balises. Cette contrainte introduit une hiérarchie entre les éléments définis par les balises. Par exemple, l'élément <ADRESSE>100, rue Mouffetard</ADRESSE> est un sous-élément de l'élément défini par la balise <CINEMA>. Ce dernier englobe tout le document (sauf la première ligne) et est appelé *l'élément racine* du document.

On peut noter que le nom des balises, ainsi que l'imbrication de ces balises, sont totalement libres : il n'existe pas en XML de règles prédéfinies. Cela permet à chacun de définir son propre langage pour décrire ses données. *L'Épée de bois* s'est défini un langage basé sur le vocabulaire `CINEMA`, `NOM`, `ADRESSE` et `METRO`, et une règle de construction très simple consistant à imbriquer dans <CINEMA> les trois autres éléments. Nous parlerons le plus souvent de « dialecte » pour désigner un langage défini avec XML.

La structure hiérarchique d'un contenu XML devient plus explicite si on ajoute des changements de ligne et des espaces dans le document :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<CINEMA>
  <NOM>Epée de Bois</NOM>
  <ADRESSE>100, rue Mouffetard</ADRESSE>
  <METRO>Censier-Daubenton</METRO>
</CINEMA>
```

Cette chaîne est considérée comme identique à la première, si on met à part les caractères blancs et les sauts de ligne entre deux balises (en fait nous verrons que cette mise à l'écart des « espaces » ne va pas toujours de soi, et peut soulever quelques problèmes dans les traitements). L'indentation est souvent utilisée pour clarifier un contenu XML, sans que cela ajouter quoi que ce soit au contenu lui-même. En fait, ce que cette pratique suggère, c'est qu'une approche fructueuse pour faciliter la compréhension et le raisonnement sur une information structurée avec XML est de considérer que cette information est un arbre, et pas une chaîne de caractères. La figure 1 montre l'arbre correspondant au cinéma « L'Epée de bois ». Il a pour racine un élément <CINEMA> donc le contenu est lui-même constitué des trois éléments <NOM>, <ADRESSE> et <METRO>. Chacun de ces trois éléments a un contenu qui est du texte simple.

Cette représentation, plus abstraite, permet de mettre l'accent sur les deux aspects vraiment essentiels d'un contenu structuré avec des balises XML. En premier lieu elle montre quels sont les noms d'éléments qui décomposent et désignent les différentes parties du contenu. En second lieu elle permet de situer précisément la place de chaque élément au sein de la hiérarchie globale. Il est très important de noter dès maintenant que ces deux aspects sont indissociables : *un élément est caractérisé à la fois par son nom et par sa place dans l'arbre XML*. Concrètement, cela implique que tout traitement de données XML – par exemple en vue de les mettre en forme pour une publication – se base sur des outils permettant de choisir des éléments par leur nom, ou par leur position, ou par les deux dimensions utilisées simultanément.

Voici un exemple plus général illustrant la structure d'un document. Il développe la description du cinéma « L'Epée de bois ». Ce cinéma a deux salles, chacune représentée par un élément XML de type

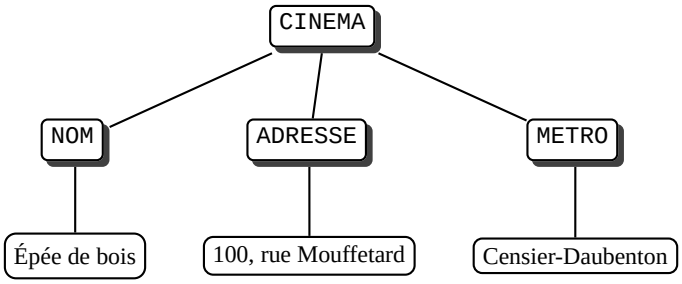

FIG. 1 – Représentation arborescente d'un contenu XML

<SALLE>. Pour chaque salle on connaît le titre du film projeté et les horaires des séances. La première salle a également un sous-élément de type <REMARQUE> qui est absent pour les deux autres salles.

»>Logbook error: File '../EXEMPLES/ExXML1.xml' does not exist!

Cet exemple introduit un nouvel aspect de la représentation d'une information en XML : les *attributs*. Dans les premiers exemples donnés précédemment, toute l'information concernant un type d'élément était codée dans le *contenu* des éléments, ou, autrement dit, dans le texte entouré par les balises de l'élément. Ici (exemple **??**), le numéro et le nombre de places disponibles pour chaque salle ne font pas partie du contenu de l'élément correspondant, mais sont indiqués au sein des balises même sous forme *d'attributs* NO et PLACES. La même information aurait pu être codée avec des sous-éléments. Ce choix entre l'utilisation d'un attribut ou d'un élément pour stocker une information concernant un élément est souvent arbitraire : nous proposerons dans le chapitre **??** des considérations pour guider ce choix.

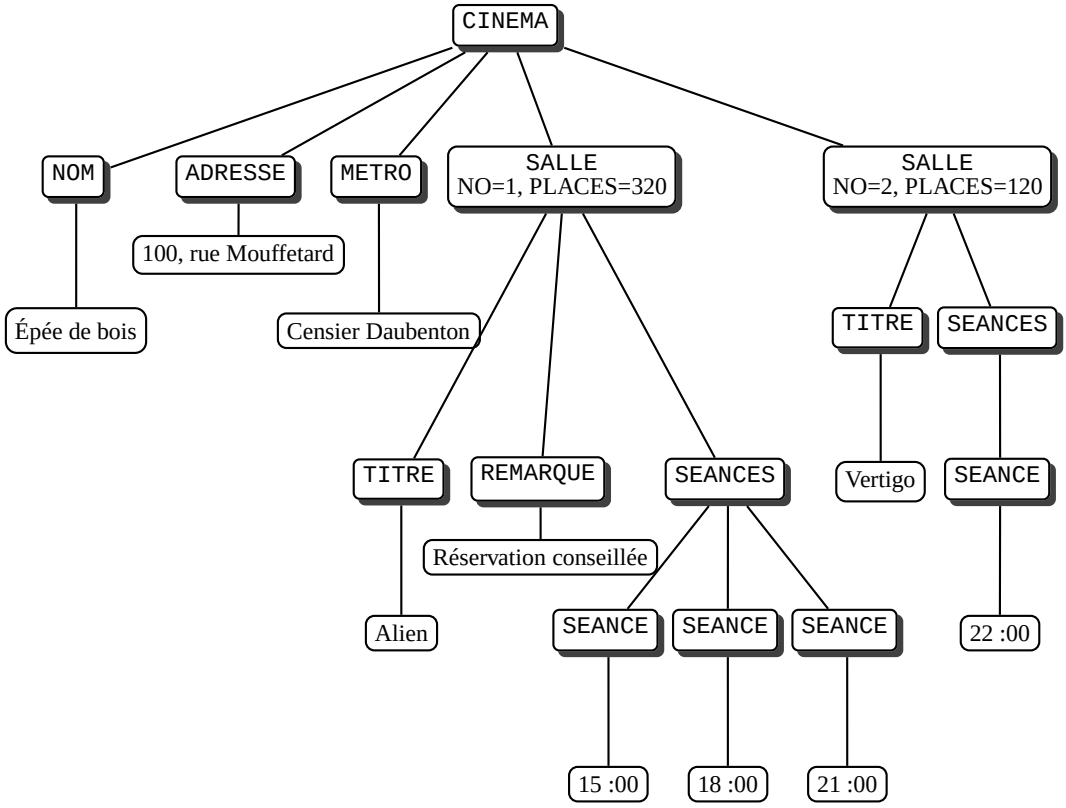

FIG. 2 – Arbre d'un document XML complété

La figure 2 montre l'arbre XML. On peut en tirer quelques constatations générales :

- cet arbre n'est pas équilibré (certaines branches sont plus longues que d'autres) ;
- certains types d'éléments se répètent (cas des salles et des séances) et d'autres pas (cas du nom du cinéma) ;
- certains types d'éléments apparaîssent régulièrement comme sous-élément d'un autre type d'élément (cas des salles qui sont des sous-éléments des cinémas et des titres qui sont des sous-éléments des salles) ;
- les éléments de certains types sont optionnels (cas des remarques).

Nous utiliserons abondamment dans la suite de ce livre la représentation arborescente pour soutenir nos explications sur les mécanismes de transformation XSLT. Elle permet de s'affranchir de détails inutiles (comme, par exemple : d'où vient cette chaîne ? comment est-elle indentée ?) et de travailler sur une représentation claire et pertinente du contenu et de sa structuration.

2.3 Documents XML

Jusqu'à présent nous avons parlé de « chaîne de caractères » sans faire d'hypothèses sur l'origine ou le stockage de cette chaîne : cette dernière peut être placée dans un fichier, mais il peut également s'agir d'un flux (ou *message*) échangé entre deux programmes ou machines, ou d'une information engendrée à partir d'une application. Par exemple, la plupart des systèmes de bases de données actuels permettent de générer des documents XML à partir des données stockées.

Dans le cas d'un stockage fichier, l'information codée en XML peut être affichée et modifiée par un outil de traitement de texte standard. Il est ainsi possible de visualiser et modifier un document XML très facilement et sans outil sophistiqué. Pour des raisons de simplicité, c'est la situation que nous envisagerons prioritairement dans ce qui suit. C'est celle aussi que nous vous invitons à employer, au moins dans un premier temps, pour manipuler nos exemples.

Le caractère persistant (cas d'un fichier) ou transitoire (cas d'un message) d'une information codée avec XML sont cependant des aspects secondaires tant qu'on ne s'intéresse pas à des problèmes d'architecture sur lesquels nous reviendrons plus tard. Nous utiliserons uniformément dans ce qui suit le terme de *document XML* pour désigner un contenu structuré avec des balises XML, et ce quelle que soit la nature physique et la durée d'existence de ce contenu.

La notion de document en XML est un peu plus complète que celle d'un arbre d'éléments. La déclaration XML, ainsi que certaines autres informations qui apparaissent avant l'élément racine sont considérées comme parties intégrantes du document. Nous distinguerons donc soigneusement, à partir de maintenant :

- *l'élément racine*, défini par la première balise rencontrée ;
- *la racine du document* qui comprend, outre l'élément racine, un ensemble de déclarations et d'instructions utiles pour l'interprétation du contenu.

Cette distinction, ainsi que la différence entre document XML et fichier, sont illustrés par l'exemple suivant. Imaginons que dans notre application, chaque salle est gérée par un responsable qui doit tenir à jour les informations. Il existe alors autant de fichiers XML qu'il y a de salles. Le document relatif à la salle 2 est le suivant :

»>Logbook error: File '../EXEMPLES/Salle2.xml' does not exist!

On retrouve les attributs et les éléments <SEANCE>, et l'élément <FILM> auquel on a ajouté des informations complémentaires : année de production, metteur en scène, résumé, etc. Le contenu de ce fichier correspond à un document représenté dans la figure 3, avec la racine du document notée `Salle2.xml :/`, un commentaire XML marqué par `<!--` et `-->`, puis l'élément racine `SALLE`.

À partir de l'ensemble des documents décrivant les salles, il est possible de reconstituer un document global en assemblant dans un fichier les informations propres au cinéma, et en important les documents relatifs aux salles. On utilise la notion XML *d'entité externe* qui permet de faire référence à une source de données externe au fichier XML « principal » (appelé *entité document*), et d'inclure le contenu de cette source.

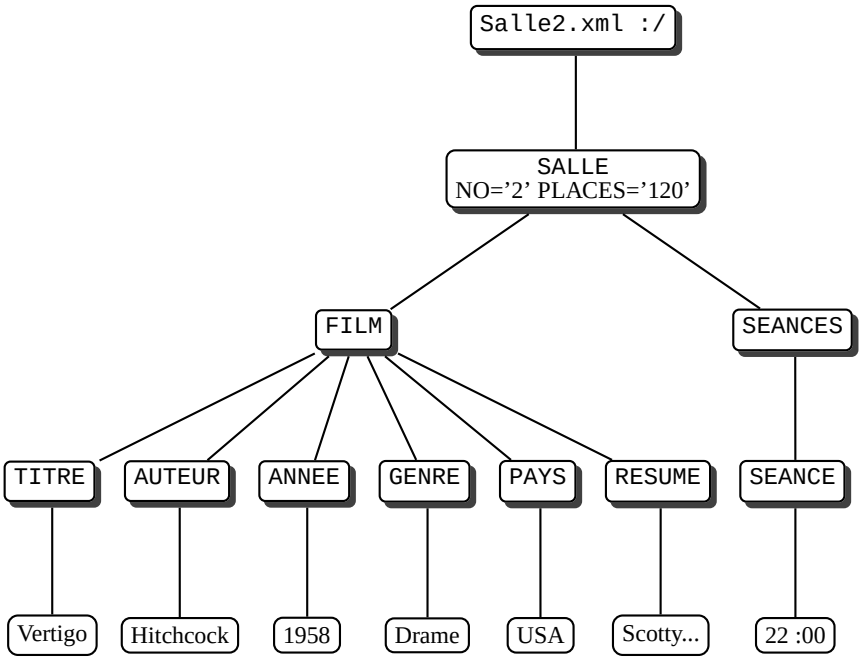

FIG. 3 – Arbre d’un document XML complété

Voici une nouvelle représentation du document XML pour le cinéma « L’Epée de bois » de la figure 2. Il s’agit bien du *même* document (avec un description plus détaillée des films), mais réparti dans trois fichiers ou entités externes :

»>Logbook error: File ’../EXEMPLES/Epee.xml’ does not exist!

L’entité document *Epee.xml* contient dans son entête deux définitions d’entités externes vers les fichiers *Salle1.xml* et *Salle2.xml* (nous verrons la signification de la balise < `!DOCTYPE`> plus loin) :

```
<!ENTITY salle1 SYSTEM "Salle1.xml">
<!ENTITY salle2 SYSTEM "Salle2.xml">
```

Les *références* vers ces entité externes t sont représentées par le nom de l’entité correspondante entouré des symboles `&` et `;` :

```
&salle1;
&salle2;
```

Pour obtenir la représentation arborescente du document (figure 2), toutes ces références doivent être remplacées par le contenu des entités référencées (*Salle1.xml* et *Salle2.xml*).

Nous serons amené dans la suite de ce livre à généraliser considérablement ce premier exemple où un document XML est obtenu par assemblage de plusieurs sources. Ici nous sommes dans un cas simple où les sources sont d’autres fichiers, mais il est possible d’intégrer des composants très divers, incluant des parties « statiques » (par exemple un ou plusieurs fichiers, éventuellement répartis sur l’Internet) et des parties « dynamiques », fragments créés à la volée, par exemple par extraction à partir d’une base de données.

3 Publication de données avec XSLT

Nous allons maintenant montrer comment, à partir d’un document XML proposant un contenu structuré, on obtient avec des transformations XSLT (acronyme de *eXtensible Stylesheet Langage Transformations*) des publications de ce contenu destinées aux supports les plus variés. L’idée de base d’un environnement XML/XSLT est de séparer le traitement des données du processus de publication.

– le *traitement des données* consiste à les mettre sous forme de document XML obéissant à une structure arborescente donnée (par exemple la structure cinéma-salle-film illustrée dans ce qui précède) ;
– la *publication des données* consiste à extraire un contenu d'un document XML et à le mettre dans un format reconnu par une application de publication particulière (par exemple au format HTML reconnu par les navigateurs web).

Dans un tel environnement les responsabilités sont clairement partagées. On y distingue ceux qui gèrent les contenus et doivent établir les documents XML, éventuellement à partir de plusieurs sources de données – nous reviendrons sur cet aspect ultérieurement – de ceux qui doivent créer une mise en forme pour tel ou tel support. Nous prenons comme exemple les documents XML de la section précédente, sans avoir besoin de faire d'hypothèses sur l'origine de ces documents. Les transformations XSLT vont permettre d'obtenir trois versions différentes de ce même contenu :

– un site web, affichant les informations sur le cinéma, ses salles et ses séances ;
– un site WAP permettant de consulter sur un téléphone mobile les mêmes informations ;
– enfin un document PDF donnant l'ébauche de ce qui pourrait être un « Officiel des Spectacles » imprimé.

3.1 Site web (HTML)

Nous prenons maintenant comme exemple un document XML restreint à la description du film *Vertigo*. Notre but est d'obtenir une représentation HTML de ces informations.

»>Logbook error: File '../EXEMPLES/Vertigo.xml' does not exist!

HTML est un langage de balisage dédié à l'échange de documents sur le Web sous forme de *document hypertextes*[2] dans lesquels certains mots ou groupes de mots (les *ancres*) servent d'accès à d'autres documents HTML, parfois situés sur d'autres sites web. L'utilisation des ancres permet de « naviguer » sur le Web sans même connaître ses notions de base (le protocole HTTP, le système d'adressage par des URL, etc).

Contrairement à XML, HTML fournit un *ensemble fixe* de balises auxquelles sont attachées des *fonctionnalités précises de présentation*. L'utilisation des balises HTML est donc avant tout destinée à donner des directives de mise en forme au navigateur qui se charge d'afficher le document. À l'opposé, l'auteur d'un document XML définit son propre vocabulaire de balises, le choix étant guidé par le besoin de décrire du mieux possible la structure et la signification des données.

Nous reviendrons en détail dans le chapitre **??** sur le langage HTML, ses différentes versions (et notamment sa variante XHTML), et ses rapports avec XML. Ici nous nous contenterons de montrer comment on peut obtenir une version HTML de nos documents XML par transformation XSLT, en commençant par prendre l'exemple très simple de la description d'un film. La figure 4 montre la page HTML que nous désirons obtenir.

On constate que le titre du film apparaît en caractères gras italiques, le pays d'origine en caractères italiques et le nom du réalisateur en caractères gras. De plus le texte entoure une image de l'affiche (version française) du film. Le document HTML qui permet d'obtenir ce résultat est le suivant :

»>Logbook error: File '../EXEMPLES/Vertigo.html' does not exist!

Il s'agit, une nouvelle fois, d'un arbre dont la racine est l'élément <`html`>, avec deux fils : <`head`> et <`body`>. Le reste du document est du texte libre, encadré par des balises de mise en forme. Sans entrer pour l'instant dans le détail, on peut :

– centrer une partie d'un document en utilisant la balise <`center`> ;
– différencier différents niveaux de titre de sections (balises <`h1`>, <`h2`>, ..., <`h5`>) ;
– mettre du texte en italique (balise <`i`>) ou en gras (balise <`b`>) ;
– créer des paragraphes (balise <`p`>) ;
– inclure des images (balise <`img`>) qui sont stockées dans des fichiers externes.

Ce document HTML peut tout à fait être vu comme un document XML, avec une structure imposée (notamment pour la racine de l'arbre et ses deux fils), et des noms de balise fixées par la norme HTML. En fait HTML s'est développé indépendamment de XML et s'avère beaucoup moins rigoureux. Les navigateurs acceptent par exemple des documents sans la racine <`html`>, et dans de nombreux cas la balise

[2] « HTML » est l'abréviation de « HyperText Markup Language ».

FIG. 4 – Version HTML du document *Vertigo.xml*, affichée par Netscape

fermante d'éléments vides est considérée comme optionnelle. Une norme récente, XHTML, reconsidère la définition de HTML comme un « dialecte » XML : nous y reviendrons au chapitre **??**.

3.1.1 Transformations XSLT

Nous allons maintenant utiliser un programme XSLT pour produire automatiquement le document *Vertigo.html* (appelé *document résultat*) à partir du document *Vertigo.xml* (appelé *document source*). Comme c'est la première fois que nous rencontrons ce langage, nous commençons par une brève introduction permettant d'en comprendre les principes de base.

L'application d'un programme XSLT consiste à parcourir les nœuds d'une arborescence XML, et à appliquer des *règles de transformation* à ces nœuds. Voici un premier exemple de règle de transformation XSLT. Elle consiste à produire une phrase simple chaque fois qu'un élément <`FILM`> est rencontré dans le document source.

```
<xsl:template match="FILM">
  Ceci est le texte produit par application de cette règle.
</xsl:template>
```

La notion de règle (nommées *template*) est centrale dans XSLT, et la bonne compréhension des aspects suivants est essentielle.

- une règle s'applique *toujours* dans le *contexte* de l'un des nœuds du document source ;
- l'application de la règle consiste à produire un fragment de document, qui peut combiner du texte et des données extraites du document source.

Une règle est un élément XML défini par la balise `xsl:template`. L'attribut `match="FILM"` est le *motif de sélection* ou *pattern* de cette balise et indique à quel(s) type(s) d'éléments du document XML traité s'applique la règle. La règle ci-dessus s'appliquera donc *toujours* dans le contexte d'un élément de type `FILM`. Enfin le contenu de l'élément `xsl:template` est le *corps* de la règle et définit le texte produit à chaque fois que la règle s'applique. Le début du chapitre **??** approfondira toutes ces notions.

Un programme XSLT est en général constitué de plusieurs règles dont chacune peut s'appliquer à différentes parties d'un document XML. Chaque règle produit un sous-arbre spécifique quand elle rencontre un élément déclencheur, et l'assemblage de ces sous-arbres constitue le résultat de la transformation. Dans le cas le plus simple, on spécifie des règles différentes pour chaque type d'élément d'un document, comme nous l'avons fait ci-dessus pour l'élément <`FILM`> du document XML à transformer. Parfois la distinction par le type de l'élément peut être insuffisante et/ou trop imprécise pour choisir les règles et des motifs de sélection plus complexes, que nous présenterons plus tard, s'avèrent nécessaires.

Le corps de la règle indique le fragmentdu résultat produit quand un nœud du bon type est rencontré dans le document XML. Dans notre premier exemple, le texte produit est toujours

```
Ceci est le texte produit par application de cette règle.
```

et ce quel que soit le contenu de l'élément <FILM>. En pratique ce genre de règle n'est évidemment pas très intéressante, et on rencontre plus souvent des règles où le texte reprend et réorganise des parties du document XML traité. Rappelons que la règle s'applique dans le contexte d'un nœud : le principe consiste alors à sélectionner, *à partir de ce nœud*, les parties du document qui vont être produites par la règle.

Prenons l'exemple du document XML représentant le film *Vertigo*. Quand notre règle s'exécute, le nœud contexte est l'élément racine de l'arbre. On a accès, depuis cette racine, à tous les fils du nœud <FILM>, soit les éléments <TITRE>, <AUTEUR>, <ANNEE>, etc., mais aussi à d'autres nœuds comme par exemple les nœuds descendants (les fils des fils, etc...) ou les nœuds parents et ancêtres.

Pour sélectionner le contenu d'un élément, on utilise *l'élément XSLT* `xsl:value-of` en indiquant simplement le chemin d'accès à partir du nœud contexte Par exemple le titre du film est obtenu par :

```
<xsl:value-of select="TITRE"/>
```

L'élément `xsl:value-of` est un exemple d'élément « vide ». Quand un élément `A` n'a pas de contenu, au lieu d'écrire <A></A>, on utilise une seule balise <A/> qui est à la fois ouvrante et fermante. En général les informations associées à un tel élément sont représentées sous forme d'attribut(s), comme par exemple l'attribut `select` pour `xsl:value-of`.

La figure 5 montre l'interprétation de `xsl:value-of`, avec ses deux paramètres déterminants : le contexte d'exécution, qui est ici l'élément <FILM>, et le *chemin d'accès*, en l'occurrence l'élément <TITRE>, fils de l'élément-contexte. Il s'agit ici de l'exemple le plus simple – mais aussi le plus courant – où on accède aux descendants directs d'un élément. Nous verrons que le mode de désignation des éléments à partir d'un nœud contexte est beaucoup plus général.

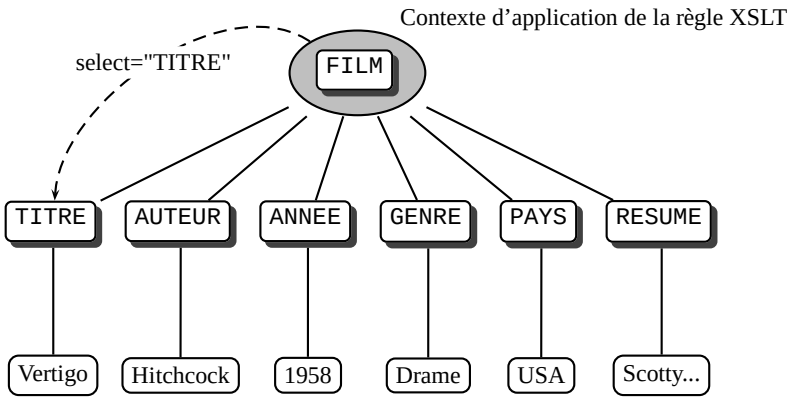

FIG. 5 – Extraction des informations dans une règle XSLT

Nous sommes maintenant en mesure d'enrichir notre première règle pour produire un premier document HTML sommaire, mais complet.

```
<xsl:template match="FILM">
  <html>
    <head>
      <title>Film:
          <xsl:value-of select="TITRE"/>
      </title>
    </head>
    <body>
```

```
      Ici je peux mettre d'autres informations sur le film
    </body>
  </html>
</xsl:template>
```

L'élément <`xsl :value-of select="TITRE"/`> est remplacé à l'exécution par la valeur de l'élément <`TITRE`>, soit `Vertigo`. Appliquée à *Vertigo.xml*, cette règle produira donc le résultat suivant :

```
<html>
  <head>
    <title>Film:
        Vertigo
    </title>
  </head>
  <body>
    Ici je peux mettre d'autres informations sur le film
  </body>
</html>
```

Bien entendu on peut appliquer la même règle à tout document XML ayant la même structure que *Vertigo.xml*.

3.1.2 Transformation XML -> HTML

Notre objectif est donc d'obtenir le document HTML de l'exemple **??**, page **??**, par transformation XSLT du document *Vertigo.xml*, page **??**. De manière générale le résultat est obtenu en insérant dans des balises HTML des extraits du document XML. Tous ces extraits sont facilement accessibles à partir de la racine <`FILM`>, et nous allons donc pouvoir nous contenter d'une seule règle qui va « piocher », à partir de la racine, les différents constituants décrivant le film, et produire les fragments HTML appropriés.

La règle de transformation, insérée dans un programme complet *Film.xsl*, crée un document HTML pour chaque film, autrement dit pour chaque document XML ayant la même structure que *Vertigo.xml*.

»>Logbook error: File '../EXEMPLES/Film.xsl' does not exist!

Un programme XSL comme celui présenté ci-dessus est un document XML interprété par un processeur[3] XSLT : on peut vérifier que c'est un arbre constitué d'une imbrication de balises ouvrantes et fermantes. Il doit obéir à certaines règles de structuration et de nommage des balises. En particulier toutes les balises propres à XSLT doivent être préfixées par `xsl :` avec que le processeur puisse les distinguer des balises à insérer dans le résultat (dites *éléments littéraux*). Ce préfixe définit un *espace de noms* qui caractérise les balises XSLT (les espaces de noms et leur prise en compte avec XSLT constituent un aspect particulier qui sera abordé dans le chapitre **??**).

L'élément racine du document XSLT *Film.xsl* est de type `xsl:stylesheet` et contient deux sous-éléments de type `xsl:output` et `xsl:template`. L'élément `xsl:output` informe le processeur XSLT que le document généré est un document HTML (cette information est nécessaire pour des raisons qui seront expliquées dans le chapitre **??**).

[3] Nous désignons par ce terme le programme chargé d'effectuer la transformation.

Remarque : Vous pouvez dès maintenant récupérer les fichiers sur notre site et expérimenter par vous-mêmes les transformations si vous avez installé les outils proposés dans l'annexe **??** (ou d'autres). Avec Xalan par exemple la transformation s'effectue sur la ligne de commande :

```
java org.apache.xalan.xslt.Process -in Vertigo.xml
                    -xsl Film.xsl -out Vertigo.html
```

Un fichier *Vertigo.html* est alors créé. Une autre possibilité (à utiliser par exemple avec le processeur XSLT de Internet Explorer) est d'inclure directement une instruction au début du document XML pour indiquer le programme XSLT à appliquer :

```
<?xml-stylesheet href="Film.xsl" type="text/xsl"?>
```

La transformation s'effectue alors « dynamiquement ». Voir l'annexe **??** pour plus de détails.

La figure 6 illustre l'application de cette règle par deux arbres reliés par des arcs en pointillés : l'arbre inférieur correspond au document XML qui doit être transformé en HTML (pour une meilleure représentation, la racine de l'arbre est en bas) ; l'arbre supérieur correspond à la règle contenant la structure de la page HTML à générer. Tous les nœuds sauf sept feuilles sont étiquetés par des balises HTML qui définissent la structure du document produit. Les autres noeuds sont de type `xsl:value-of` et définissent, pour leur part, la partie *dynamique* du contenu, obtenue par extraction de certains éléments du document XML. On peut noter que l'élément <TITRE> est référencé trois fois et que l'ordre des instructions de substitution ne correspond pas à l'ordre des éléments insérés dans le résultat.

3.1.3 Chemins complexes

Nous prenons maintenant l'exemple de la production du document HTML décrivant une salle de cinéma pour illustrer la capacité de XSLT à spécifier des chemins d'accès complexes pour extraire des informations d'un document XML.

Dans le document HTML montrant un film, tous les nœuds <TITRE>, <AUTEUR> et autres auxquels on accédait était les *fils* (au sens de : descendants directs) de l'élément <FILM> tenant lieu de contexte de la règle. Le cas d'un document décrivant une salle de cinéma présente une structure plus riche. La figure 7 montre la salle 1 de notre cinéma. Si on prend comme contexte d'une règle de transformation le nœud <SALLE>, on constate que :

- certaines informations (le numéro de la salle, le nombre de places) sont représentées comme des *attributs* ;
- le nœud a trois fils : <FILM>, <REMARQUE> et <SEANCES> ;
- les informations décrivant le film sont des petits-fils de <SALLE> : <TITRE> et <AUTEUR> en sont séparés par <FILM> ;
- l'élément <SEANCE> est petit-fils de <SALLE>, et peut de plus être répété un nombre quelconque de fois ;
- enfin l'élément <REMARQUE> est optionnel : il apparaît pour la salle 1, mais pas pour la salle 2 (voir figure 3, page 7).

Ces nouvelles particularités rendent plus complexe l'accès aux informations permettant de créer un document HTML à partir d'un nœud de type SALLE. Gardons pour l'instant le principe de n'utiliser qu'une seule règle faisant appel à des `xsl:value-of` . Les parties du document *Salle1.xml* sont alors désignées par des *chemins* qui prennent tous leur origine du nœud <Salle>. Voici quelques exemples, illustrés dans la figure 7.

- l'attribut d'un élément est désigné par la concaténation du symbole '@' et du nom de l'attribut ; donc le numéro de la salle est désigné par `select="@NO"` ;
- le descendant d'un élément est désigné en donnant les nœuds successifs rencontrés à chaque niveau de la branche menant à l'élément ; donc le titre du film est désigné par `select="FILM/TITRE"` ;

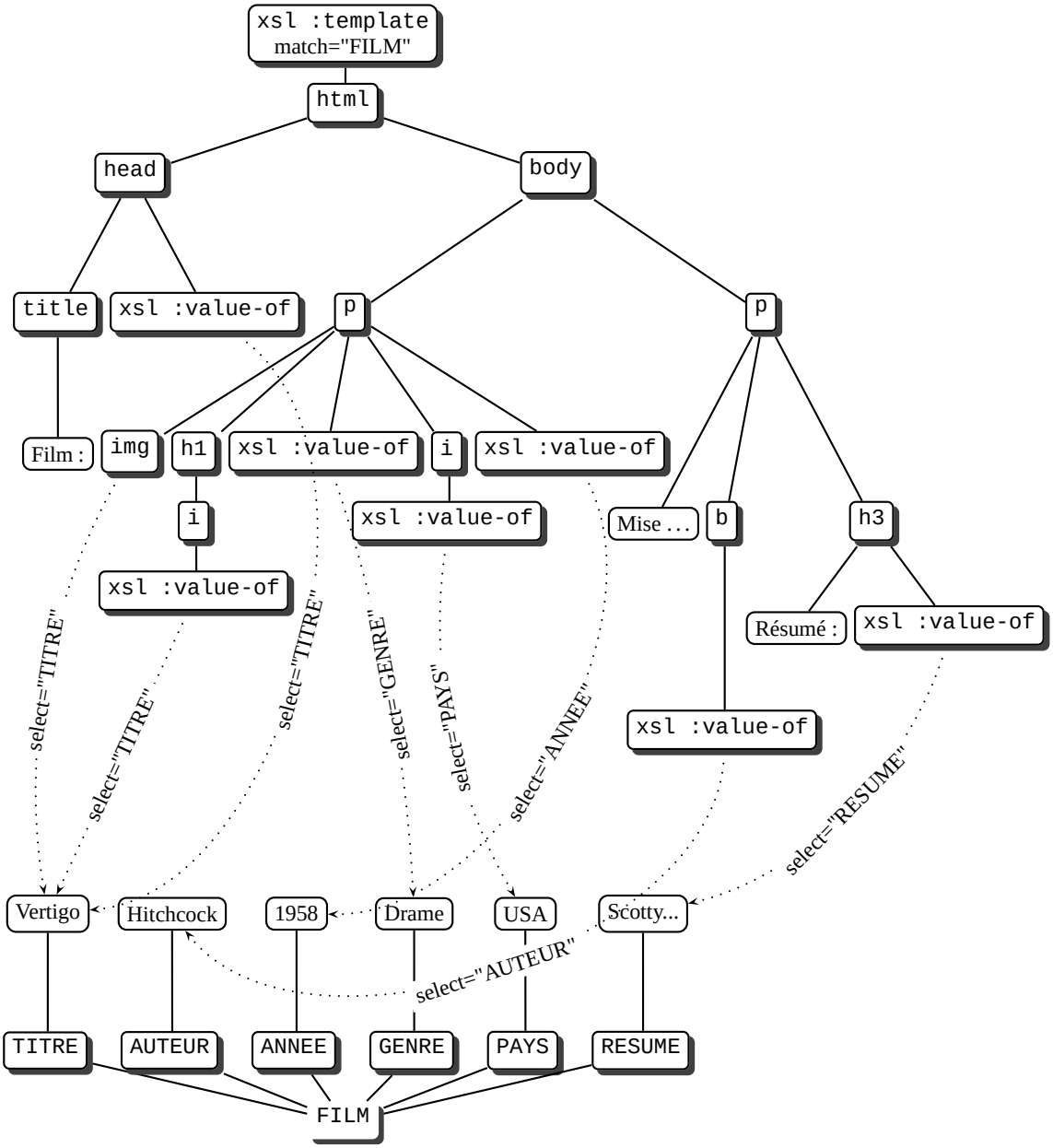

FIG. 6 – Transformation XSLT

FIG. 7 – Le document XML pour la première salle

– pour parcourir toutes les occurrences d'un même type d'élément, on peut effectuer une *boucle* `xsl:for-each` sur l'ensemble des séances désignées par `select="SEANCES/SEANCE"` ;
– enfin l'élément contexte lui-même est désigné par le symbole « . », comme dans `select="."`.

Voici la règle produisant une présentation HTML et récupérant les informations en exprimant les chemins présentés ci-dessus.

```
<xsl:template match="SALLE">
 <!-- Extraction d'attribut : no de salle et places  -->
 <h2>Salle No  <xsl:value-of select="@NO"/></h2>
 <h3>Capacité: <xsl:value-of select="@PLACES"/> places</h3>

 <!-- Description du film: tous les chemins commencent par FILM/ -->
 <h3>Film:     <a href="{FILM/TITRE}XML.html">
   <b>         <xsl:value-of select="FILM/TITRE"/>
 </b></a></h3>
 de <b>        <xsl:value-of select="FILM/AUTEUR"/>
 </b><br/><i>  <xsl:value-of select="FILM/PAYS"/></i>,
               <xsl:value-of select="FILM/ANNEE"/>

 <!-- Boucles sur toutes les séances                    -->
 <h3>Séances</h3>
  <ol>
   <xsl:for-each select="SEANCES/SEANCE">
     <li>      <xsl:value-of select="."/></li>
   </xsl:for-each>
 </ol>
 <xsl:for-each select="REMARQUE">
    <font color="red"><xsl:value-of select="."/></font>
 </xsl:for-each>
</xsl:template>
```

L'instruction `xsl:for-each` permet de créer une *boucle* pour ajouter toutes les séances et remarques dans le résultat. La transformation définie par le contenu d'un élément <`xsl :for-each`> ne s'applique pas à l'élément qui est en cours de transformation (la salle), mais aux éléments sélectionnés par

l'attribut `select` (les séances). Ainsi, l'instruction <`xsl :value-of select="."/`> dans la première boucle sera remplacé par le contenu de chaque séance retrouvée.

3.1.4 Appel de règle

Il reste à ajouter les balises <`html`>, <`head`> et <`body`> pour obtenir un document HTML complet. Jusqu'à présent nous nous sommes volontairement limités à des exemples de productions basés sur une seule règle. En pratique on est le plus souvent amené à définir *plusieurs* règles dans un programme XSLT, et à déclencher des appels de règles à partir d'autres règles. La figure 8 montre la page HTML que l'on veut obtenir telle qu'elle s'affiche avec Netscape.

FIG. 8 – Copie écran du résultat de la transformation de *Salle1.xml*

La règle initiale s'applique en général à la racine du document XML, désignée par «/». Elle génère un document HTML complet avec un élément racine <`html`> et deux sous-éléments <`head`> et <`title`>. Cette règle est la *première* règle appliquée par le processeur XSLT pendant la transformation d'un document XML.

```
<xsl:template match="/">
  <html>
    <head>
       <title>
          Salle: <xsl:value-of select="SALLE/@NO"/>
       </title>
    </head>
    <body bgcolor="white">
      <xsl:apply-templates select="SALLE"/>
    </body>
  </html>
</xsl:template>
```

L'élément <`body`> ne contient qu'un seul sous-élément qui est une instruction XSLT de type `xsl:apply-templates`. Le processeur XSLT remplace cet élément par le *résultat de la transformation* qu'il déclenche. Autrement dit on va appliquer aux éléments <`SALLE`> fils de la racine «/» du document la règle spécifique à ce type d'élément, et insérer entre les balises <`body`> et <`/body`> le fragment HTML produit.

L'intérprétation du mécanisme d'appel de règles est donné dans la figure 9. La partie supérieure montre les deux règles du programme XSLT, et la partie inférieure l'arbre XML à transformer.

– La première règle (en bas à gauche dans la figure) s'applique à la racine du document, «/». Elle produit successivement un élément <`html`>, racine de l'arbre résultat, puis deux fils, <`head`> et <`body`>.

FIG. 9 – Appel de règles à partir d’autres règles

– Au sein de l'élément <body> se trouve une instruction d'application d'une règle aux éléments fils de l'élément courant qui sont de type SALLE. Le processeur XSLT cherche donc si un tel élément existe : c'est le cas et la deuxième règle (en bas à droite) est donc déclenchée. Elle produit des fragments HTML qui viennent s'insérer dans l'élément <body>.

3.1.5 Réutilisation de règles

FIG. 10 – La page d'accueil de *L'Épée de bois*

Passons maintenant à un dernier stade pour compléter cette introduction à XSLT. Nous voulons produire la page HTML récapitulant tout le programme proposé par « L'Épée de bois ». La figure 10 montre le résultat obtenu[4]. Il apparaît clairement que la présentation de chaque salle est *identique* à celle qui était obtenue par application de la règle destinée à présenter une salle individuelle.

Pour créer la présentation d'un cinéma, nous allons donc reprendre la règle existante de traitement des éléments SALLE, et déclencher son application. On utilise pour cela un élément `xsl:apply-templates`. Voici tout d'abord la règle s'appliquant à l'élément racine du document XML.

```
<xsl:template match="/">
  <html>
    <head>
      <title>
        Programme de <xsl:value-of select="CINEMA/NOM"/>
      </title>
    </head>
    <body bgcolor="white">
```

[4] Il va sans dire que nous privilégions ici la simplicité du code, au détriment de la qualité de présentation.

```
      <xsl:apply-templates select="CINEMA"/>
    </body>
  </html>
</xsl:template>
```

La seconde règle d'applique dans le contexte d'un élément <CINEMA>. Elle extrait des informations relatives au cinéma, et propage la production de code HTML par un nouvel appel à `xsl:apply-templates`.

```
<xsl:template match="CINEMA">
  <h1>
    <i>
      <xsl:value-of select="NOM"/>
    </i>
  </h1>
  <hr/>
  <xsl:value-of select="ADRESSE"/>,
  <i>Métro: </i>
  <xsl:value-of select="METRO"/>
  <hr/>

  <xsl:apply-templates select="SALLE"/>

</xsl:template>
```

Ce dernier appel déclenche la règle relative aux salles, pour tous les éléments <SALLE> qui sont fils de l'élement <CINEMA>. Nous ne redonnons pas cette dernière règle.

FIG. 11 – L'arbre HTML, résultat de la transformation XSLT

L'application de chaque règle engendre un fragment d'arbre HTML, et tout appel `xsl:apply-templates` insère dans ce fragment un ou plusieurs sous-arbres. La figure 11 montre (partiellement) l'arbre final HTML obtenu, en distinguant les parties issues de chacune des trois règles impliquées dans la transformation.

1. la première règle, appliquée à la racine du document XML, produit un « squelette » de document HTML, avec les trois balises <html>, <head> et <body> indispensables, puis elle passe la main à la règle s'appliquant aux éléments de type CINEMA ;
2. la seconde règle produit les informations globales sur un cinéma, puis déclenche l'application de la règle aux éléments <SALLE> ;
3. enfin la troisième règle s'applique aux deux éléments <SALLE> présents dans la description du cinéma *L'Épée de bois*, et produit les deux sous-arbres HTML décrivant ces deux salles.

Un programme XSLT peut donc être vu comme l'application successive d'un ensemble de règles, chacune concourant à la création du résultat final par « assemblage » de fragments qui viennent s'associer pour former une hiérarchie. Dans la copie d'écran du résultat de la transformation (figure 10) on peut clairement distinguer la partie créée par la règle de transformation pour les salles, reproduisant un motif de présentation identique pour chaque élément <SALLE> rencontré.

3.2 Site WAP (WML)

Passons maintenant à la création d'une représentation permettant de consulter les séances à partir d'un téléphone mobile, en partant du même document XML décrivant les séances à « L'Epée de bois ».

3.2.1 Publication WAP (WML)

Le langage utilisé pour spécifier l'affichage sur un mobile est WML. Il s'agit d'un « dialecte », ou spécialisation de XML, qui se base sur un ensemble de noms d'éléments et définit leur signification.

Un document WML est un arbre avec pour racine un élément <wml> (de même qu'un document HTML est un arbre avec pour racine l'élément <html>), mais il est divisé en sous-éléments, les « cartes », définisant l'unité d'affichage sur l'écran du mobile. Les cartes d'un même document peuvent être reliées entre elles par des liens, ce qui permet à l'utilisateur de naviguer grâce aux boutons de son téléphone.

Un exemple de document WML comprenant une seule carte est donné ci-dessous, avec les informations sur le film *Alien*. Toutes les balises WML sont en minuscules, et toutes sont très concises afin de limiter le coût des transferts de documents.

»>Logbook error: File '../EXEMPLES/Alien.wml' does not exist!

FIG. 12 – L'affichage de la carte de l'exemple **??**

Une carte est organisée en paragraphes grâce à la balise <p>, avec au moins un paragraphe par carte, ce qui est le cas dans l'exemple ci-dessus. Quelques autres balises, reprises de HTML, peuvent être utilisées pour définir une mise en page (limitée) du texte sur l'écran du mobile : la balise <b> affiche le contenu

de l'élément en gras, et la balise
 permet de passer à la ligne. La figure 12 montre l'affichage obtenu dans un téléphone Nokia 6150.

Voyons maintenant comment obtenir cette représentation WML à partir du document *Alien.xml*, donné ci-dessous.

»>Logbook error: File '../EXEMPLES/Alien.xml' does not exist!

Nous allons simplement utiliser deux règles. Une première règle, d'initialisation, crée le « squelette » du document résultat, soit l'équivalent pour WML de la structure <html>, <head>, <body> que nous avons utilisée pour HTML. Cela consiste :

– à ouvrir, puis refermer la balise principale <wml> ;

– à déclencher, dans la balise principale, un appel aux autres règles.

Voici cette première règle.

```
<xsl:template match="/">
  <wml>
    <xsl:apply-templates/>
  </wml>
 </xsl:template>
```

La seconde règle s'applique à un élément de type <FILM>. Cette règle produit une carte WML, et place dans cette carte un extrait des éléments constituant la description du film, à savoir le titre, l'année, l'auteur et le résumé. Notons que l'ordre dans lequel on utilise ces éléments peut être totalement reconsidéré par rapport à celui du document XML initial.

```
<xsl:template match="FILM">
  <card>
      <p>
      <b> <xsl:value-of select="TITRE"/></b>,
          <xsl:value-of select="ANNEE"/>,
          <xsl:value-of select="AUTEUR"/><br/>
          <xsl:value-of select="RESUME"/><br/>
     </p>
   </card>
</xsl:template>
```

Voici finalement le programme complet. Il va sans dire qu'il s'applique à tout document XML dont la structure est identifique à celle de *Alien.xml*, par exemple *Vertigo.xml* (voir page **??**).

»>Logbook error: File '../EXEMPLES/FilmWML.xsl' does not exist!

L'application de ce programme permet d'obtenir le document de l'exemple **??**, et donc l'affichage de la figure 12.

3.2.2 Création d'un site WAP

Nous pouvons maintenant, à partir de notre document XML décrivant *L'Epée de bois*, créer un ensemble de cartes permettant de choisir la salle et le film, de consulter les séances pour cette salle, et enfin d'afficher les informations détaillées sur le film. La structure du document WML résultat est donnée dans la figure 13. Il contient cinq éléments <card> : un pour le menu d'accueil, montrant les salles de *L'Epée de bois* et le film qui y est projeté, deux cartes pour les deux salles, enfin deux cartes donnant des détails sur les films.

Pour permettre la navigation dans ce document, chaque carte doit être identifiée par la valeur – unique au sein du document WML – de l'attribut id dans la balise <card>. La figure 13 montre les id de chaque carte : index, S1, S2, Alien et Vertigo. Pour créer un lien vers une carte, on utilise en WML, comme en HTML, la balise <a>. Voici par exemple l'élément WML permettant de référencer la carte décrivant *Alien* :

```
<a href="#Alien">Film : Alien</a>
```

FIG. 13 – L'arbre XML du site WAP

En se positionnant sur le texte tenant lieu de lien, l'utilisateur peut afficher la carte correspondante. Les flèches en tiretés sur la figure indiquent la navigation entre les cartes : de l'index (affichage dans la figure 14, à gauche), on peut se rendre sur une salle (affichage dans la figure 14, au centre), puis d'une salle sur le film projeté dans la salle (figure 14, à droite).

Voici maintenant quelques détails sur le programme XSLT permettant de produire le document WML. La règle d'initialisation procède comme précédemment, en produisant un élément <wml>, puis elle déclenche trois autres règles, correspondant respectivement aux trois types de cartes : liste des salles, liste des séances, affichage du film. Voici cette première règle :

```
<xsl:template match="/">
  <wml>
    <!-- création de la carte d'accueil                -->
    <xsl:apply-templates select="CINEMA"/>

    <!-- création des cartes pour les salles et séances -->
    <xsl:apply-templates select="CINEMA/SALLE"/>

    <!-- création des cartes pour les films            -->
    <xsl:apply-templates select=".//FILM"/>
  </wml>
```

Rappelons comment on désigne le chemin d'accès aux éléments auxquels on applique des règles. Le principe tient en deux points :

1. prendre en compte la position courante (le *nœud contexte*) dans l'arbre ;
2. indiquer le chemin à suivre dans l'arbre, *à partir du nœud contexte*, pour atteindre l'élément à traiter.

Ici la position courante à laquelle la règle s'applique, autrement dit le nœud contexte de la règle, est la racine du document, dénotée « / ». On veut traiter trois types d'éléments, <CINEMA>, pour produire la carte d'index, <SALLE>, pour produire la carte donnant les séances, et <FILM> pour produire la carte donnant la description de chaque film.

Étant donnée la position courante située à la racine, et la position de ces éléments dans l'arbre XML, la désignation de ces éléments se fait de trois manières différentes :

1. l'élément <CINEMA> est un fils de la position courante : on le désigne simplement par CINEMA ;
2. les éléments de type SALLE sont petit-fils de la position courante, avec comme niveau intermédiaire <CINEMA> : on les désigne par CINEMA/SALLE qui indique le chemin à parcourir ;

FIG. 14 – Un téléphone mobile avec carte WML

3. enfin les éléments de type `FILM` sont arrière-petit-fils de la position courante ; on pourrait les désigner par `CINEMA/SALLE/FILM`, mais on utilise ici un moyen plus général : `.//FILM` désigne tous les éléments descendant de la position courante, quel que soit leur niveau, nommés <`FILM`>.

Une part importante de la programmation XSLT consiste à savoir désigner, à partir d'un nœud courant dans l'arbre XML, les éléments auxquels on veut appliquer des règles de transformation. Les quelques exemples donnés ci-dessus correspondent à des cas restreints, et font notamment l'hypothèse que le chemin d'accès à un élément se fait en descendant dans l'arbre. Nous verrons que le langage de chemin utilisé, nommé XPath, est beaucoup plus général.

Il reste à créer une règle pour chacun des types de carte. Voici la règle, déclenchée sur les éléments de type `CINEMA`, qui produit la carte affichant la liste des salles.

```
<xsl:template match="CINEMA">
  <card id="index" title="Programme">
   <p align="center">
    <xsl:value-of select="NOM"/>
   </p>
   <xsl:for-each select="SALLE">
     <p> <a href="#S{@NO}">
          Salle <xsl:value-of select="@NO"/>:
         </a>
           <xsl:value-of select="FILM/TITRE"/>
     </p>
   </xsl:for-each>
  </card>
</xsl:template>
```

Rappelons que tout ce qui ne commence pas par « `xsl :` », indiquant les types d'éléments propres

à XSLT, est inclus dans le résultat. Pour un cinéma, on va afficher avec une boucle `xsl:for-each` un paragraphe (élement de type `p`) pour chaque salle. Ce paragraphe contient l'intitulé de la salle, et sert également d'ancre (élément <a>) vers la carte détaillant les séances de la salle. La carte d'une salle est référencée par `#S`x où x est le numéro de la salle, présent dans l'attribut `NO`. Avec XSLT, on crée cet identifiant avec l'expression `#S{@NO}`.

Voici maintenant la règle servant à produire les cartes des salles. Notez que chaque carte comprend un attribut `id` ayant pour valeur `S`x où x est le numéro de la salle. On insère une ancre vers la carte détaillant le film diffusé dans la salle.

```
<xsl:template match="SALLE">
  <card id="S{@NO}">
   <p>Séances salle  <xsl:value-of select="@NO"/></p>
   <p>
      <a href="#{FILM/TITRE}">
        Film : <xsl:value-of select="FILM/TITRE"/>
      </a>
    </p>
   <xsl:apply-templates select="SEANCES"/>
  </card>
</xsl:template>
```

Enfin la règle produisant la carte des films est identique à celle que nous avons déjà étudiée (exemple **??**, page **??**).

3.3 Document papier

Chaque type de document a des contraintes spécifiques – affichage, mode d'utilisation – définies par son interface utilisateur. L'auteur doit prendre en compte la taille de la fenêtre d'affichage, la couleur des pages, la couleur et la taille des caractères, la possibilité – ou non – d'effectuer un défilement, etc. Ces critères sont très stricts par exemple pour un téléphone portable dont l'écran est très petit et les manipulations limitées par un clavier réduit à une quinzaine de touches.

La création d'un document destiné à l'impression est essentiellement différente de ceux destinés au Web et constitue un changement radical de support. Une page HTML est affichée sur un écran informatique et doit avoir une taille limitée, approximativement égale à la taille de l'écran. De plus la lecture de documents HTML n'est pas fondée sur un parcours linéaire, de bas en haut et de gauche à droite, mais sur une « navigation » à travers des liens. Par contraste, un document papier est organisé en une séquence de pages, avec un ordre de lecture séquentielle. En conséquence, quand on imprime un document HTML, le résultat est souvent insatisfaisant. Nous complétons cette présentation de l'utilisation de XSLT pour la transformation de documents XML avec la production de documents papier grâce aux *formatting objects*.

3.3.1 Les *formatting objects* (XSL-FO)

Le langage XSL-FO est, comme (X)HTML ou WML, un « dialecte » XML destiné à décrire la mise en forme d'un document résultat. XSL-FO est, historiquement, lié à XSLT et en constitue un complément : le projet initial visait à définir un langage de « feuilles de style » pour documents XML, et s'est rapidement divisé en deux parties, l'une traitant des transformations (XSLT) et l'autre de la présentation (XSL-FO).

Le couple XSLT/XSL-FO permet d'appliquer un processus complet de publication en deux étapes (figure 15) :

1. l'étape de *transformation* avec XSLT permet d'extraire des informations du document source et de les « marquer » avec des balises XSL-FO pour indiquer la présentation souhaitée : dans le cas d'un document papier, la présentation prend en compte la taille des pages, les marges à respecter, la taille et le poids des caractères, l'alignement du texte, l'espace entre les paragraphes, l'en-tête et le pied de page, etc ;
2. l'étape de *mise en forme* (*formatting* en anglais) s'applique au document XSL-FO obtenu précédemment, et consiste à créer un document final adapté au support de publication, dans un format standard comme PDF ou Postscript.

FIG. 15 – Transformation et Mise en Forme

XSL-FO est une recommandation officielle du W3C (*FO* est un acronyme pour *formatting objects*). Le langage permet la spécification précise des caractéristiques typographiques d'un document. Signalons que nous lui consacrons un chapitre complet, le **??**, et que le lecteur pressé peut donc sauter sans dommage les explications qui suivent.
Voici un premier document XSL-FO :

»>Logbook error: File '../EXEMPLES/SimpleFO.xml' does not exist!

Le langage apparaît assez rébarbatif, mais rappelons qu'il s'agit en principe d'un document intermédiaire produit par une transformation XSLT (personne n'est donc obligé de le créer à la main...). Voici quelques explications sur ce document. Ce document a pour élément racine `fo:root`, le préfixe `fo :` indiquant au processeur XSL-FO quelles sont les balises correspondant à des instructions de mise en forme. On trouve ensuite quelques éléments qui indiquent les propriétés globales de la mise en page souhaitée :

- l'élément <`fo :simple-page-master`> qui définit la dimension (`page-height="29.7cm"`, `page-width="21cm"`) des pages ;
- l'élément <`fo :region-body`> qui fixe l'espace entre la « région » du contenu et les bords de chaque page (`margin-top="2cm"`, `margin-bottom="2.5cm"`, ...) ;
- l'élément <`fo :page-sequence-master`> qui est utilisé pour définir la structure du document entier ou d'une partie (par exemple : une table des matières, des chapitres, un index, etc) ;
- enfin l'élément <`fo :block`> qui définit des paragraphes avec l'espace de séparation (`space-before="20pt"` et la taille des caractères (`font-size="10pt"`).

Les éléments qui suivent correspondent au contenu du document. Ce dernier contient deux paragraphes (éléments de type `fo :block`) dont le deuxième est séparé du premier par un espace de 20 points (`space-before="20pt"`). La taille des caractères est 20 points par défaut (`font-size="20pt"` dans <`fo :flow`>) sauf pour le deuxième paragraphe, où la taille des caractères est limité à 10 points.

Le document obtenu peut maintenant être transmis à un processeur XSL-FO qui va se charger de produire une réalisation concrète. Nous utilisons le processeur FOP qui produit des documents PDF : il va nous permettre d'engendrer une version imprimable du programme de *L'Épée de bois*. (figure 16).

3.3.2 Le programme de *L'Épée de bois*

Voyons maintenant comment on produit une version PDF – un « tract » – présentant le programme de notre cinéma, toujours à partir du même document XML *Epee.xml*, page **??**. On va définir deux règles de transformation. La première définit la présentation du nom, de l'adresse et de la station de métro du cinéma.

```
<xsl:template match="CINEMA">
  <fo:block text-align="center">
    <fo:block font-size="40pt" font-weight="bold" space-after="20pt">
      <!-- Sélection du nom du cinéma       -->
      <xsl:value-of select="NOM"/>
    </fo:block>
```

FIG. 16 – Mise en Forme avec XSL-FO

```
    <!-- Sélection de l'adresse du cinéma   -->
    <xsl:value-of select="ADRESSE"/>
    <!-- Sélection du métro près du cinéma     -->
    (<xsl:value-of select="METRO"/>)
    </fo:block>
    <fo:block space-before="20pt">
    <!-- Transformer chaque salle -->
    <xsl:apply-templates select="SALLE"/>
  </fo:block>
</xsl:template>
```

Le nom du cinéma et son adresse sont des sous-blocs d'un bloc spécifiant que le texte sera centré (attribut `text-align="center"`). Le nom du cinéma est écrit en caractères gras de 40 points, suivi de l'adresse et de la station de métro. Pour afficher les séances, la règle suivante est appelée (élément <xsl :apply-templates select="SALLE"/>) :

```
<xsl:template match="SALLE">
  <fo:block text-align="center" space-before="40pt">
    <fo:inline font-weight="bold" font-size="26pt">
```

```
      <!-- Sélection du titre du film   -->
      <xsl:value-of select="FILM/TITRE"/>
    </fo:inline> de
    <fo:inline font-style="italic">
      <!-- Sélection de l'auteur du film   -->
      <xsl:value-of select="FILM/AUTEUR"/>
    </fo:inline>
    <fo:block space-before="5pt">
      <!-- Sélection de l'année et du pays du film   -->
      (<xsl:value-of select="FILM/PAYS"/>,
       <xsl:value-of select="FILM/ANNEE"/>)
    </fo:block>
  </fo:block>
  <fo:block space-before="10pt">
    <!-- Sélection du résumé du film   -->
    <xsl:value-of select="FILM/RESUME"/>
  </fo:block>
  <fo:block space-before="10pt">
    <fo:inline font-weight="bold">
      <!-- Sélection du numéro de salle   -->
      Salle <xsl:value-of select="@NO"/>
    </fo:inline>
    (<xsl:value-of select="@PLACES"/> places) :
    <!-- Sélection des séances   -->
    <xsl:for-each select="SEANCES/SEANCE">
      <xsl:value-of select="."/> -
    </xsl:for-each>
    <!-- Sélection de la remarque   -->
    <xsl:value-of select="REMARQUE"/>
  </fo:block>
</xsl:template>
```

Ces deux règles effectuent la transformation de la structure du document XML et la définition de la présentation du résultat sous forme d'un document XSL-FO.

En résumé, à partir d'un même document XML, nous sommes en mesure de produire avec un seul langage des représentations très diverses. Nous allons passer maintenant à un contexte d'utilisation légérement différent de XSLT en étudiant les possibilités d'intégration et d'échanges de données offertes par XML.

4 Échange et intégration de données en XML

La technologie XML n'est pas limitée à la génération de pages web (HTML, WAP) ou d'autres documents plus traditionels. Par sa capacité de représenter des données très diverses, XML a rapidement été promu comme format universel pour l'échange de données entre différéntes applications informatiques.

Dans cette section nous décrivons l'utilisation de XML comme interface entre plusieurs applications souhaitant échanger des données. Notre exemple va montrer l'intérêt de XML comme format d'échange entre serveurs web, mais il s'extrapole facilement à l'utilisation de XML pour échanger des informations sous forme de *messages* entre des applications réparties.

Nous mettons plus l'accent sur le processus de transformation impliqué dans ces échanges que sur la transmission des données qui est liée à l'infrastructure client-serveur et aux protocoles de communication.

4.1 Exemple : Le Site *www.sallesenligne.com*

L'utilisation de XML comme format d'échange est illustrée par un site web qui récupère des informations sur les programmes de cinéma disponibles sur différents sites locaux, les intègre et fournit un moteur de recherche. La Figure 17 montre le site *www.sallesenligne.com*, relié à de nombreux sites locaux dont deux, *www.cine-marseille* et *www.epee-de-bois.fr*, sont représentés.

FIG. 17 – Échanges de données du site *www.sallesenligne.com*

La recherche d'un film et d'une séance de cinéma se base sur le titre du film, le début de la séance et la ville. L'utilisateur remplit un formulaire avec les critères désirés et obtient une liste des séances correspondantes avec des liens vers les sites locaux (Figure 18).

FIG. 18 – Formulaire de saisie et résultat fourni par le moteur de recherche

Le site *www.sallesenligne.com* offre donc une valeur ajoutée aux informations déjà existantes au niveau de chaque site élémentaire. En contrepartie, il demande que les informations nécessaires à la recherche lui soient fournies sous un format XML imposé dont voici un exemple :

```
<FILM>
  <TITRE>Vertigo</TITRE>
  <CINEMA>Epée de Bois</CINEMA>
  <VILLE>Paris</VILLE>
  <URL>www.epee-de-bois.fr/ExCinema1.xml</URL>
  <HEURE>22:00</HEURE>
</FILM>
```

À part le titre, le cinéma, la ville et les heures de projection, l'élément <FILM> contient un sous-élément de type URL avec l'adresse de la page web du cinéma. Ces informations forment un résumé de toutes celles disponibles au niveau du site d'un cinéma, et elles sont de plus représentées différemment, aussi bien au niveau de la structure des documents que du nom des balises utilisées. Le but est donc de convertir les documents XML vers la structure définie pour le moteur de recherche, de transférer cette

conversion vers *www.sallesenligne.com* et enfin d'assembler tous ces fragments dans un seul document sur lequel s'effectueront les opérations de recherche.

4.2 Description de la structure d'un document XML

Pour écrire un programme de transformation, il faut connaître la structure du document à transformer, mais également la structure du document à engendrer. Comme dans les exemples sur la publication, où la structure du résultat d'une transformation devait être conforme aux « formats » HTML, WML ou XSL-FO, l'information exploitée par le moteur de recherche doit « ressembler » à la structure de l'élément <FILM> précédent.

En l'occurrence, le serveur central veut acquérir des éléments de type FILM, avec des sous-éléments de type TITRE, CINEMA, VILLE, URL et HEURE. Chaque sous-élément n'apparaît qu'une seule fois sauf les sous-éléments de type HEURE.

Cette description informelle de la structure des documents XML échangés peut être utilisée pour écrire un programme XSLT qui génère des « résumés » à partir des documents stockées sur les sites locaux. Pour éviter les malentendus et ambiguités qui sont possibles dans un langage naturel, il existe des langages plus formels pour décrire la structure d'un document XML. Nous introduisons ci-dessous la méthode la plus répandue à l'heure actuelle pour définir des types de documents : les *Document Type Definitions* ou *DTD* (ce sujet fait l'objet d'un développement complet dans le chapitre **??**).

Dans une DTD, la structure d'un document XML est spécifiée par la description structurelle des types d'éléments. Ainsi, à travers une DTD, un élément n'est plus seulement caractérisé par le nom de la balise, mais également par la structure de son contenu. Par exemple, l'expression

```
<!ELEMENT TITRE ( #PCDATA ) >
```

indique que le contenu (aussi appelé *modèle de contenu*) d'un élément de type TITRE est une chaîne de caractères (#PCDATA). Il en est de même d es éléments de type CINEMA, VILLE, URL et HEURE. Le contenu des éléments de type FILM est en revanche défini par une expression plus complexe :

```
<!ELEMENT FILM (TITRE, CINEMA, VILLE, URL?, HEURE+)
```

Cette expression indique que le contenu d'un élément <FILM> est constitué d'une succession d'éléments dont les types respectifs sont TITRE, CINEMA, VILLE, URL et HEURE. La séparation des noms d'éléments par des virgules signifie que l'ordre des éléments est fixé et doit correspondre à la séquence indiquée. Tous les types d'éléments ne peuvent apparaître qu'une seule fois comme fils d'un élément <FILM>, sauf l'élément de type URL qui est optionel (ce qui est indiqué par le symbole ?) et les éléments de type HEURE qui peuvent apparaître plusieurs fois (indiqué par le symbole + après HEURE). Voici la DTD complète, spécifiant les informations acceptées par le moteur de recherche :

»>Logbook error: File '../EXEMPLES/Echange.dtd' does not exist!

Il est possible d'indiquer, dans l'entête d'un document XML, à quelle DTD se conforme le contenu du document. Cette information, quoique non obligatoire, est très importante car elle définit la classe des applications qui vont pouvoir interpréter et traiter le document. Réciproquement, une application (par exemple : un navigateur) est conçue pour traiter la classe de tous les documents XML conformes à une DTD donnée (par exemple la DTD XHTML).

Un programme peut également être considéré comme une transformation dédié à une DTD. C'est ce que nous avons exprimé – de manière peu précise – en signalant qu'un programme de transformation du document *Alien.xml* s'applique également à *Vertigo.xml*. De plus, si le résultat est toujours un document XML conforme à une autre DTD, un programme XSLT peut être vu comme une conversion de documents XML entre deux structures bien définies.

Les exemples de publications que nous avons décrits (HTML, WML, XSL-FO) étaient conformes à ce cadre, puisque pour chacun de ces langages il existe une DTD. Maintenant, XML étant un langage *eXtensible*, chacun peut définir sa propre DTD, et la publication XSLT s'étend à la mise en forme de nos données afin qu'elles soient reconnues et traitées par quelqu'un d'autre. Le cas que nous traitons n'est donc qu'une généralisation naturelle d'une problématique de publication qui se limiterait à HTML, ou aux quelques langages largement diffusés.

4.3 Transformation et échange de données

L'application – le site central – traite des documents conformes à la DTD *Echange.dtd*, et les intègre dans une liste formant un *index* de toutes les séances de cinémas. La structure de ce document est très simple : l'élément racine de type `MOTEUR` contient un sous-élément <`FILM`> pour chaque film projeté dans un cinéma. Notez dans l'en-tête la référence à la DTD *Moteur.dtd* qui décrit la structure du document.

»>Logbook error: File '../EXEMPLES/Moteur.xml' does not exist!

Afin d'obtenir un résumé du document *EpeeDeBois.xml* qui puisse être intégré dans le moteur de recherche, il reste à lui appliquer une transformation à l'aide du programme suivant.

»>Logbook error: File '../EXEMPLES/Echange.xsl' does not exist!

Le mécanisme de transformation devrait maintenant être clair pour le lecteur. Notons que cette transformation peut intervenir soit au niveau de chaque cinéma, qui fait alors l'effort de fournir un programme de transformation de ses données en escomptant en tirer profit, soit au niveau du moteur de recherche lui-même.

4.4 Un moteur de recherche XML/XSLT

Pour conclure cet exemple, nous allons décrire l'implantation d'un petit moteur de recherche pour trouver des séances de cinéma dans un document XML. Nous supposons que toutes les informations sont stockées dans *Moteur.xml*. Le programme suivant publie *toutes* les séances sous forme d'un document HTML.

»>Logbook error: File '../EXEMPLES/MoteurHTML.xsl' does not exist!

Il peut facilement être adapté pour sélectionner un sous-ensemble des films à publier en utilisant des *paramètres*. En l'occurrence, on peut introduire trois paramètres de sélection `$titre`, `$seance` et `$ville` dans le programme, comme suit :

```
<xsl:param name="titre"/>
<xsl:param name="seance"/>
<xsl:param name="ville"/>
```

Ces déclarations globales sont ajoutées après l'élément <`xsl :output method="html"/`> et avant la première règle de transformation. Il suffit ensuite de modifier la règle de transormation pour les éléments de type `MOTEUR` en ajoutant une condition de sélection avant la transformation d'un film :

```
<xsl:template match="MOTEUR">
  <xsl:for-each select="FILM">
    <xsl:if test=" ($titre = ''  or TITRE = $titre)
                and ($seance = '' or HEURE &gt;= $seance)
                and ($ville = ''  or VILLE = $ville)">
      <xsl:apply-templates select="." /><p/>
    </xsl:if>
  </xsl:for-each>
</xsl:template>
```

L'instruction `xsl:if` compare le contenu des éléments de type `TITRE`, `HEURE` et `VILLE` avec les paramètres correspondants. Si la condition est satisfaite, l'élément de type `FILM` est transformé. Dans le cas opposé, la boucle `xsl:for-each` passe au film suivant.

Les valeurs des paramètres formels `$titre`, `$seance` et `$ville` peuvent être obtenues de différentes manières. Dans le cas d'une application web, on utilise naturellement un formulaire HTML (figure 17) qui passe les paramètres dans l'URL du programme (méthode `GET` du protocole HTTP). On obtient par exemple :

```
http://wwww.sallesenligne.fr/Moteur.xsl?titre=Vertigo&ville=Paris
```

En résumé, les programmes XSLT interviennent ici à deux niveaux : pour la *transformation* des documents issus de chaque site, en vue d'une uniformisation de la stucture des données telle qu'elle est définie par la DTD du moteur de recherche, et pour *l'extraction* sélective de parties du document intégrateur à

l'aide de paramètres. Ce dernier aspect montre que XSLT est *aussi* un langage de programmation assez généraliste qui permet d'effectuer des boucles, des tests et de manipuler des variables. On ne peut cependant pas comparer XSLT avec un langage de programmation classique (comme le C ou Java), et nous verrons dans le chapitre **??** les limites qui sont rapidement rencontrées dès que l'on se confronte à des algorithmes un tant soit peu complexes.

4.5 Intégration dynamique de fragments XML

Dans tous les exemples que nous avons étudiés jusqu'à présent les informations étaient issues de sources XML, éventuellement transformées au passage par XSLT. Supposons maintenant qu'une partie des données soit déjà présente dans le système d'information, stockées non pas sous forme de document XML mais, par exemple, dans une base de données relationnelle. Se pose alors le problème d'extraire de la base les données, de les mettre au format XML, en les intégrant éventuellement avec celles qui sont en XML « natif », enfin d'appliquer une transformation sur le tout.

Nous allons donc conclure ce tour d'horizon – partiel – des possibilités de publication à partir de XML/XSLT en étudiant la situation où la représentation XML est produite *dynamiquement* à partir d'une autre source de données. Il s'agit de l'un des sujets les plus avancés de ce livre, qui sera traité dans le chapitre **??**. La présentation qui suit vise donc essentiellement à donner un aperçu des difficultés soulevées par cette situation.

La notion de dynamicité est maintenant bien établie dans le domaine des sites web. Les sites constitués de pages HTML au contenu figé sont en effet extrêmement difficiles à maintenir, et tout à fait inappropriés pour interagir avec un internaute qui vient visiter le site, y rechercher des informations, voire déposer ses propres données. La plupart des sites professionnels sont donc associés à une base de données qui gère l'information indépendamment de toute représentation, les données étant extraites et mises en forme « à la volée » par des techniques de programmation spécialisées basées sur l'interface CGI (*Common Gateway Interface*). Les principaux langages dédiés à ce type d'application sont Perl, PHP et Java, ce dernier sous la forme de *servlets* ou de leur dérivé, les *Java Server Pages* (JSP).

Dans le cas de PHP ou des JSP, le principe consiste à intégrer dans un document HTML des parties programmées qui vont se connecter à la base de données (ou à toute autre source d'information), effectuer des requêtes SQL et formater le résultat de ces requêtes avec des balises HTML. Il est tout à fait souhaitable de pouvoir faire de même avec XML, dans la mesure où on admet qu'un système d'information ne gére pas *toutes* ses données sous forme XML, mais fait appel à des applications spécialisées : traitement de texte, tableur, annuaire, et bien sûr Systèmes de Gestion de Bases de Données (SGBD).

Notons que PHP ou les JSP sont – du moins à l'heure actuelle – conçus pour être utilisés uniquement avec HTML. Le scénario que nous considérons ici est plus général, et se déroule en deux étapes. Dans un premier temps l'organisme ou la société qui souhaite publier une partie du contenu de son système d'information intégre ce contenu sous une forme homogène XML. Puis, à partir de cette intégration les différents types de présentation étudiés précédemment (HTML, WML, PDF, etc) sont générés par des programmes XSLT.

Nous prenons dans ce qui suit l'exemple d'une intégration dynamique (autrement dit au moment où le document est utilisé) à partir d'une base de données relationnelle. Cette base contient une table *Film* dont nous allons extraire une ou plusieurs lignes.

titre	auteur	année	genre	pays	résumé
Alien	Ridley Scott	1979	USA	Science-Fiction	Près d'un vaisseau spatial échoué sur une lointaine planète, des Terriens en mission découvrent de bien étranges "oeufs"...
Vertigo	Alfred Hitchcock	1958	Drame	USA	Scottie Ferguson, ancien inspecteur de police, est sujet au vertige depuis qu'il a vu mourir son collègue...
...	...	...	...	...	...

XML n'est pas un langage de programmation et ne permet donc pas directement d'exécuter des requêtes SQL pour extraire les données de cette table. Il n'existe pas – pas encore ? – de norme pour définir comment des parties programmées peuvent être intégrées dans du XML, mais le schéma généralement suivi est celui illustré dans la figure 19.

FIG. 19 – Architecture d'une intégration XML/Base de données

1. des balises « dynamiques », interprétées par l'application comme correspondant à l'exécution de code d'accès à une base de données, sont insérées dans le document initial ;
2. avant la transformation par une feuille XSLT, ces balises déclenchent le code qui va extraire des informations et en faire un fragment XML ;
3. finalement ce fragment vient remplacer les balises « dynamiques » : un document intermédiaire est obtenu, qui peut être publié avec XSLT.

Nous utilisons en l'occurrence les outils suivants : MySQL pour le SGBD, Java pour le code de connexion et d'interrogation de la base, enfin XSP, un langage issu du projet Cocoon de la fondation Apache qui reprend le principe des JSP et permet d'exécuter du code Java inclus dans un document XML. Le mécanisme peut être transposé, *mutatis mutandis*, à d'autres environnements, par exemple l'environnement XML proposé par Oracle qui propose sensiblement le même schéma (voir chapitre **??**).

L'exemple développé ci-dessous va extraire la description XML d'un film et l'intégrer dans le document XML correspondant aux séances de *L'Épée de bois*.

4.5.1 D'une base relationnelle à XML

Voici pour commencer le code Java qui permet d'accéder à la base MySQL, d'effectuer la requête SQL recherchant un film (ici c'est *Alien*) et de le mettre au format XML. Le petit programme ci-dessous utilise l'interface JDBC (*Java Database Connectivity*) pour se connecter à MySQL.

»>Logbook error: File '../EXEMPLES/ExJDBC.java' does not exist!

Les commentaires de ce programme devraient suffire à donner une compréhension, ne serait-ce qu'intuitive, de son fonctionnement. Une requête (au sens large : interrogation ou mise à jour) correspond à un objet de la classe `Statement`. Cet objet doit avoir été créé par un objet `Connection`, ce qui le rattache automatiquement à l'une des transactions en cours.

La méthode `executeQuery`, comme son nom l'indique, exécute une requête (d'interrogation) placée dans une chaîne de caractères. Le résultat est placé dans un objet `ResultSet` qui, comme son nom l'indique encore une fois, contient l'ensemble des lignes du résultat.

Un objet `ResultSet` correspond à la notion de *curseur* employée systématiquement dans les interfaces entre un langage de programmation et SQL. La classe `ResultSet` propose un ensemble de méthodes `get***` qui prennent un numéro d'attribut en entrée et renvoient la valeur de cet attribut. L'exécution de ce programme donne le résultat suivant :

»>Logbook error: File '../EXEMPLES/ExJDBC.xml' does not exist!

On est donc en présence d'une information transitoire, qu'il est possible d'intégrer à un document statique *avant* d'effectuer la transformation XSLT.

4.5.2 Intégration Java/XML/XSLT

Maintenant que les films sont dans une base de données, il devient inutile de placer la description d'*Alien* ou *Vertigo* dans des *fichiers* XML : il est bien préférable de créer à la demande la version transitoire XML, puis d'appliquer le programme existant.

Voici le nouveau document décrivant *Alien*, les éléments statiques étant remplacés par du code Java qui les engendre dynamiquement.

»>Logbook error: File '../EXEMPLES/AlienXSP.xml' does not exist!

Il s'agit d'un document XML, avec un élément racine de type `xsp :page` et une nouvelle classe d'éléments XSP dont toutes les balises étant préfixées par `xsp :`. On retrouve à peu de chose près la structure du programme Java de l'exemple **??**, avec, successivement :

1. l'appel aux *packages* nécessaires, obtenu avec le type d'élément `xsp :include` ;
2. l'inclusion directe de code Java dans les éléments de type `xsp :logic` ;
3. enfin la production avec le éléments de type `xsp :expr` de chaînes de caractères qui viennent s'intégrer au document XML.

Le document est traité en deux phases. Dans une première, le code Java est évalué et remplacé par les chaînes de caractères produites, ce qui revient en l'occurrence à obtenir un fichier – temporaire – décrivant le film *Alien*. Dans la seconde phase le programme est apppliqué et on obtient la page HTML.

Le lecteur familier avec les *Java Server Pages* (JSP) notera que XSP s'en inspire largement : dans les deux cas on inclut du code dans un langage de balisage, en limitant les parties programmées aux endroits où des informations dynamiques doivent être insérées. La grande différence est qu'une page JSP mélange HTML, Java, et même SQL. Ici on a séparé le contenu (en XML) de la présentation (obtenue par un programme XSLT).

4.5.3 Séparation des points de vue

Il reste cependant un aspect insatisfaisant dans cette séparation des rôles : en incluant directement du code Java dans du XML, on mélange toujours la gestion du contenu et la « logique » de l'application. Autrement dit la personne qui est en charge de définir les informations à publier et leur structure, est également confrontée au code qui permet de produire ces informations. Idéalement ces deux points de vues devraient également être séparés. C'est ce que permet l'utilisation de balises « dynamiques ». En voici un exemple :

»>Logbook error: File '../EXEMPLES/AlienXSPLib.xml' does not exist!

Cette fois plus aucun code n'apparaît dans le document qui est réduit à une extrême simplicité. Une nouvelle classe de balises, `BDFilm`, a été définie pour servir d'interface avec la base de données. La balise

```
<BDFilm:film titre='Alien'/>
```

exprime de la manière la plus simple qui soit une demande d'insertion de la représentation XML d'*Alien*. Par un mécanisme propre au serveur d'application, et que nous ne décrirons pas ici, le processeur va associer cette balise au code Java/JDBC présenté précédemment, constituer la représentation XML temporaire du film, puis appliquer la transformation XSLT.

On obtient, avec cette architecture, une séparation complète des rôles :

1. la *logique* de l'application est codée en Java (ou tout autre langage reconnu par le serveur), ce qui autorise tous les calculs, accès aux bases de données, échanges réseaux et appels de services divers et variés ; l'information obtenue est rendue disponible par l'intermédiaire d'une librairie de balises (*taglib*) ;
2. le *contenu* est intégré sous forme XML, à partir de sources d'informations qui peuvent être des fichiers, des sites, ou les balises des librairies ;
3. enfin la présentation est obtenue par un ensemble de programmes XSLT.

5 Comment lire la suite de ce livre ?

Le moment est venu de récapituler ce premier chapitre, et de voir comment il fournit la clé pour appréhender le contenu de ce livre, et donc en guider la lecture.

5.0.4 Récapitulatif

En utilisant XML, nous rendons notre contenu indépendant du format propre à une application donnée. Du même coup, nous dissocions ce contenu de toute signification intrinsèque. Un document XML n'a pas d'autre interprétation que celle qu'un traitement particulier va lui associer à un moment donné. Prenons deux exemples pour bien clarifier cette notion :

1. quand un traitement de texte sauvegarde un document dans son format propriétaire, ce format a une signification, définie par le rendu qu'en fait le traitement de texte lors d'un affichage à l'écran ; il n'y a rien de tel avec XML puisqu'un document est indépendant de toute application ;
2. dans une page HTML, on trouve également des balises, mais la différence essentielle est que la signification de ces balises est fixée, dans un contexte donné à l'avance qui est l'affichage dans la fenêtre d'un navigateur : une balise
 correspond à un saut de ligne, et ne s'interprète pas librement.

Ce qui compte en XML c'est la structure, à savoir l'arbre correspondant à un document, et l'utilisation de noms d'éléments comme moyen pour différencier ou au contraire grouper de nœuds. En revanche le choix des noms lui-même est sans importance et on aurait tout aussi bien pu utiliser pour décrire notre cinéma les balises <A>, <B>, <C> et <D> ou les balises <MOVIE>, <NAME>, <ADDRESS> et <SUBWAY> sans modifier la structure.

Chacun est donc libre de définir, en fonction de ses besoins, son propre langage basé sur XML (XML est parfois présenté comme un *méta-langage* – un langage pour définir des langages). C'est ce que nous avons fait pour décrire nos séances de cinéma, c'est ce que font également les navigateurs web avec (X)HTML, les mobiles avec WML, etc. Tous ces langages, dialectes de XML, partagent des règles de structuration communes, ce qui permet de leur appliquer des outils standards (édition, analyse) et de passer de l'un à l'autre sans difficulté majeure avec XSLT.

5.0.5 Échange et intégration avec XML

Le rôle de XML comme langage d'échange et d'intégration de données découle des caractéristiques qui précèdent. Dans beaucoup de cas, un système d'information confie ses données à des logiciels spécialisés dans une tâche donnée (serveur web, base de données, fichiers de configuration, etc) et les rend du même coup impropres à une utilisation autre que celle à laquelle elles ont été initialement affectées.

Transformer ses données au format XML, temporairement ou définitivement, revient à les rendre disponibles pour d'autres applications et d'autres utilisateurs. Intégrer ces informations, issues de sources diverses, dans un format commun, permet à la fois d'en donner une vision uniforme, et d'éviter de confronter le responsable de la publication à des logiciels et langages divers et complexes. De fait la connaissance de XSLT (et bien sûr du dialecte XML de sortie, HTML ou autre) suffit.

5.0.6 Publication XML/XSLT

Il existe des applications qui sont conçues pour travailler sur des documents XML ayant une structure particulière, et utilisant une terminologie (un nommage des nœuds) spécifique. De telles applications définissent la signification de tels documents, en fournissant un mise en forme (sur écran ou sur papier) ou un mode de traitement (insertion dans une base de données, envoi de messages) de leur contenu.

L'exemple typique est fourni par un document HTML – ou, pour être plus précis, XHTML – qui obéit à des règles de structuration et de nommage particuliers. Un document XHTML *est* un document XML, mais doit impérativement avoir pour racine <html>, cette racine ayant elle-même des sous-éléments <head> et <body> (dans cet ordre), etc. Tout document conforme aux règles de structuration XHTML peut être affiché dans un navigateur web qui interprète les balises du documents comme des commandes de mise en forme à l'écran.

XHTML n'est qu'un des exemples possibles de « dialecte » XML, et la présentation de XML comme un « super-HTML » ou comme un HTML « extensible » est selon nous une source de confusion. XHTML ne constitue qu'un des nombreux exemples de format de données décrit par un langage plus général et universel, XML. En d'autres termes, XML peut être considéré comme un outil pour définir des formats de données qui sont liés à des applications spécifiques. Nous verrons dans ce livre plusieurs exemples de dialectes XML, dont WML pour l'affichage sur des téléphones mobiles, RSS, une norme de description de documents utiles pour les annuaires de liens, SMIL, un format dédié aux présentation multimédia, XSL-FO, un langage de mise en forme de documents, etc. Ces dialectes de XML sont définis par des *Document Type Definition* (DTD).

À partir d'un document XML structuré dans un dialecte donné, il est possible d'effectuer des transformations du contenu afin d'obtenir des représentations spécialisées qui peuvent alors être traitées par les applications. Il existe de nombreuses techniques et outils pour effectuer de telles transformations. La perspective adoptée par ce livre est celle d'une transformation avec des programmes XSLT. Un programme applique une conversion à un document XML. Il est bien entendu possible de définir plusieurs programmes pour un même document, ce qui aboutit finalement à un mécanisme de publication de contenu illustré dans la figure 20.

FIG. 20 – Intégration XML, et transformations XSLT

Cette figure décline quelques-unes des situations qui seront explorées dans les prochains chapitres. Pour un même document XML, on obtient par des transformations XSLT :

1. une ou plusieurs pages XHTML constituant un site web ;
2. un formatage adapté à un traitement de texte, en vue de faire de l'édition ;
3. des « cartes » WML pour communiquer par le WAP avec des téléphones mobiles ;
4. enfin on peut imaginer toute conversion d'un dialecte XML vers un autre, l'objectif dans ce cas étant de faire communiquer des applications en vue de réutiliser des informations existantes.

En résumé, XML est un langage de description de données qui vise à l'universalité en se basant sur un principe simple : le contenu est séparé de la présentation, et la manière dont ce contenu est constitué est elle-même indépendante de la production d'une présentation. On obtient un processus de publication en plusieurs étapes, chacune relevant d'une compétence particulière, ce qui constraste avec les techniques

courantes de productions de site qui impliquent la maîtrise et l'emploi simultané d'un grand nombre de techniques et d'outils.

5.0.7 Organisation des chapitres qui suivent

La suite de ce livre reprend de manière systématique tous les thèmes abordés dans ce qui précède. Le chapitre **??** donne une description complète de la syntaxe XML, et introduit le langage XPath pour désigner des fragemnts dans un document. Les chapitres **??**, **??** et **??** traitent successivement des règles et de la programmation XSLT, de la production de documents XML, et du rôle tenu par XSLT dans les processus d'échange et d'intégration de données. Les chapitres qui suivent sont plus « appliqués » et décrivent l'utilisation de XSLT dans plusieurs situations typiques : interprétation d'une DTD et style de programmation (chapitre **??**), interaction entre XSLT et XSL-FO pour la production de documents papier (chapitre **??**), enfin publication de bases de données (chapitre **??**).

Nous espérons que la lecture de ce premier chapitre aura permis de clarifier les choix d'organisation qui ont été faits, et donneront l'opportunité au lecteur, en fonction de ses compétences et centres d'intérêt, de suivre éventuellement un ordre de lecture différent.