Introduction à XML

Georges-André Silber

6 janvier 2003

Table des matières

Ce document va vous permettre d'acquérir les fondements du langage de balisage extensible (XML) et des technologies qui lui sont associées (XPath, XSLT, DTD).

Les références normatives de XML se trouvent sur le site du World Wide Web Consortium (W3C) à l'adresse `http://www.w3.org`. Le site `http://www.cri.ensmp.fr/~silber/cours/xml` contient les exemples et les documents liés à ce cours. Une connaissance de base de HTML est requise pour la compréhension de ce cours.

1 Introduction

> « *Le langage de balisage extensible (eXtensible Markup Language, XML) est un sous-ensemble de SGML [...]. Son but est de permettre au SGML générique d'être transmis, reçu et traité sur le Web de la même manière que l'est HTML aujourd'hui. XML a été conçu pour être facile à mettre en œuvre et interopérable avec SGML et HTML.* »
>
> Traduction française de « Langage de balisage extensible (XML) 1.0 », Recommandation du W3C, 10 février 1998.

XML (eXtensible Markup Langage) est une notation permettant de décrire un langage, sous la forme d'une grammaire d'arbre. XML est plus général et uniforme qu'HTML et plus simple que SGML (Standard General Markup Langage) dont il est un descendant direct.

Ce formalisme a été défini par le World Wide Web Consortium (W3C) et regroupe sous l'acronyme XML une boîte à outils extensible dont le but est de simplifier la gestion (c'est-à-dire l'organisation, l'échange et la transformation) des données. Il est important de noter qu'XML défini un ensemble de règles pour organiser les jeux de données, mais aussi un ensemble de technologies permettant de travailler avec ces données organisées.

Depuis le premier document de travail de la spécification XML datant de 1998 jusqu'aux documents actuels, XML a fait couler beaucoup d'encre et de salive, surtout pour vanter ses vertus. XML n'est cependant pas la solution miracle à tous les problèmes mais simplement un outil qui, s'il est utilisé correctement, peut rendre la vie plus facile.

1.1 Un petit peu d'histoire

XML n'est pas vraiment une nouveauté puisqu'il est un sous-ensemble de SGML (Standardized General Markup Langage) modifié pour une utilisation orientée vers le WEB. SGML a été originellement développé par Goldfarb, Mosher et Lorie chez IBM en 1969, comme outil de structuration de documents légaux. Il a évolué pour devenir un standard international de représentation de données textuelles dans un format indépendant du système. Comme SGML est un peu trop complexe pour être utilisé sur le Web, XML est une version édulcorée de SGML, adapté spécialement au Web.

On peut se demander pourquoi le besoin d'un nouveau langage s'est fait sentir alors que HTML est censé être le langage universel du Web. Et bien, alors que HTML est un très bon outil pour composer des pages Web, il n'offre aucune possibilité de description des données contenues dans ces pages. HTML n'est qu'un langage de mise en forme et ne permet pas de définir des données. La principale différence entre HTML et XML est qu'HTML comporte des balises fixes alors que XML permet la définition de n'importe quelle balise.

Le W3C voulait qu'XML soit simple et facile à utiliser. XML se doit de supporter un grand nombre d'applications, en permettant aux utilisateurs de développer leurs propres balises.

2 XML en 7 points

2.1 XML est une méthode pour mémoriser des données structurées dans un fichier texte

On entend par « *données structurées* » des éléments tels que des feuilles de calcul, des carnets d'adresses, des paramètres de configuration, des transactions financières, des dessins techniques, etc.

Les programmes qui produisent de telles données les stockent souvent sur disque, dans un format binaire ou un format texte. Ce dernier format vous permet, si nécessaire, de consulter les données sans le programme qui les a produites.

XML est un ensemble de règles, de lignes directrices, de conventions, quel que soit le nom que vous voulez leur donner, pour la conception de formats texte pour de telles données, de façon à produire des fichiers qui soient faciles à générer et à lire (par un ordinateur), qui ne soient pas ambigus, et qui évitent les pièges courants, tels que la non-extensibilité, l'absence de prise en charge de l'internationalisation/localisation et la dépendance par rapport à certaines plates-formes.

2.2 XML ressemble à HTML, mais ce n'est pas HTML

Comme HTML, XML utilise des *balises* (des mots encadrés par '<' et '>') et des attributs (de la forme `nom="valeur"`), mais alors que HTML définit la signification de chaque balise et de chaque attribut (et souvent la manière dont le texte qu'ils encadrent apparaîtra dans un navigateur), XML utilise les balises seulement pour délimiter les éléments de données et laisse l'entière interprétation des données à l'application qui les lit.

En d'autres termes, si vous voyez "`<p>`" dans un fichier XML, ne supposez pas qu'il s'agit d'un paragraphe. Selon le contexte, cela peut être un prix, un paramètre, une personne, un p (d'ailleurs, qui a dit qu'il devait s'agir d'un mot commençant par "p" ?).

2.3 XML est un texte qui n'est pas destiné à être lu

Les fichiers XML sont des fichiers texte, mais ils sont encore moins destinés à être lus par des individus que les fichiers HTML.

Ce sont des fichiers texte, car ils permettent à des experts (tels que les programmeurs) de déboguer plus facilement des applications, et en cas d'urgence, d'utiliser un simple éditeur de texte pour corriger un fichier XML endommagé.

Mais les règles des fichiers XML sont beaucoup plus strictes que celles des fichiers HTML. Une balise oubliée ou un attribut sans guillemets rendent le fichier inutilisable, alors qu'avec HTML, de telles pratiques sont souvent explicitement permises, ou au moins tolérées.

Les spécifications officielles de XML indiquent que les applications ne sont pas autorisées à essayer de deviner ce qu'a voulu faire le créateur d'un fichier XML endommagé ; si le fichier est endommagé, une application doit s'arrêter immédiatement et émettre une erreur.

2.4 XML est une famille de technologies

Il existe XML 1.0, la spécification[1] qui définit ce que sont les « balises » et les « attributs », mais autour de cette spécification, un nombre de plus en plus important de modules facultatifs ont été définis, fournissant des ensembles de balises et d'attributs ou des lignes directrices pour des tâches particulières.

C'est, par exemple, le cas de **XLink**[2], qui décrit une méthode standard pour ajouter des liens hypertextes à un fichier XML.

[1] `http://www.w3.org/TR/REC-xml`
[2] `http://www.w3.org/TR/xlink`

XPointer[3] est une syntaxe permettant de se référer à des parties d'un document XML. Ces références ressemblent à des URL, mais au lieu de pointer sur des documents du Web, elles pointent sur des éléments de données au sein d'un fichier XML. C'est une norme complémentaire à **XLink**.

XSL[4] est le langage évolué pour la définition de feuilles de style. Il est basé sur **XSLT**[5], un langage de transformation qui est également souvent utilisé en dehors de XSL, pour réorganiser, ajouter ou supprimer des balises et des attributs. XSLT utilise **XPath**[6] qui est une technologie incontournable et qui permet de spécifier des « *chemins* » dans un document XML.

Le **DOM**[7] est un ensemble d'appels de fonctions standard (API, Application Programming Interface) pour manipuler des fichiers XML (et HTML) à partir d'un langage de programmation (Java, C++, Perl, etc.).

La spécification des **espaces de nom**[8] XML décrit comment vous pouvez associer un URL à chaque balise et chaque attribut dans un document XML. L'utilisation de cet URL est toutefois du ressort de l'application qui le lit. (RDF, la norme du W3C pour les métadonnées, l'utilise pour lier chaque partie des métadonnées à un fichier définissant le type de ces données.)

Les **Schémas**[9] XML aident les développeurs à définir précisément leurs propres formats basés sur XML.

Plusieurs autres modules et outils sont disponibles ou en cours de développement. Consultez régulièrement la page des rapports techniques du W3C.

2.5 XML est bavard, mais ce n'est pas un problème

Comme XML est un format texte et qu'il utilise des balises pour délimiter les données, les fichiers XML sont presque toujours d'une taille plus importante que les formats binaires équivalents.

Il s'agit là d'une décision prise en toute conscience par les développeurs de XML. Les avantages d'un format texte sont évidents (voir le point 3 ci-dessus), et ses inconvénients peuvent être généralement compensés à un autre niveau. L'espace disque n'est plus aussi coûteux qu'auparavant, et les programmes tels que `zip` et `gzip` compressent très bien et très rapidement les fichiers. Ces programmes sont disponibles pour presque toutes les plates-formes (et ils sont généralement gratuits).

De plus, les protocoles de communication tels que les protocoles de modem et HTTP/1.1 (le protocole de base du web) peuvent compresser des données à la volée, ce qui économise de la bande passante aussi efficacement qu'un format binaire.

2.6 XML est nouveau, mais pas si nouveau que ça

Le développement de XML a commencé en 1996 et XML est une norme du W3C depuis février 1998, ce qui peut laisser supposer qu'il s'agit d'une technologie plutôt immature.

En fait, ce n'est pas une technologie très nouvelle. Avant XML, il existait SGML, développé au début des années 80, devenu norme ISO depuis 1986 et largement utilisé dans des projets de documentation de taille importante. Et il existait bien sûr HTML, dont le développement a commencé en 1990.

Les concepteurs de XML ont simplement pris les meilleures parties de SGML, profité de l'expérience de HTML, et produit une technologie qui n'est pas moins puissante que SGML, mais infiniment plus régulière et plus simple à utiliser. Certaines évolutions, cependant, peuvent être assimilées à des révolutions... Il faut également savoir que SGML est principalement utilisé pour des documentations techniques et beaucoup moins pour d'autres types de données, alors que c'est exactement l'inverse avec XML.

[3] `http://www.w3.org/TR/xptr`
[4] `http://www.w3.org/TR/xsl`
[5] `http://www.w3.org/TR/xslt`
[6] `http://www.w3.org/TR/xpath`
[7] `http://www.w3.org/DOM`
[8] `http://www.w3.org/TR/REC-xml-names`
[9] `http://www.w3.org/XML/Schema`

2.7 XML est libre de droits, indépendant des plates-formes et correctement pris en charge

En choisissant XML pour un projet, vous bénéficiez d'un ensemble important et sans cesse croissant d'outils (et il est possible que l'un d'eux remplisse déjà la fonction dont vous avez besoin !) et d'ingénieurs expérimentés dans cette technologie. Opter pour XML, c'est un peu comme choisir SQL pour des bases de données : vous devez encore construire votre propre base de données et vos propres programmes ou procédures qui la manipulent, mais un grand nombre d'outils et d'individus peuvent vous y aider.

Comme XML, en tant que technologie W3C, est libre de droits, vous pouvez vous en servir pour construire votre propre logiciel sans avoir à payer quoi que ce soit à qui que ce soit. Sa prise en charge importante, et qui ne cesse de croître, signifie également que vous n'êtes pas lié à un seul fournisseur.

XML n'est pas toujours la meilleure solution, mais vaut toujours la peine d'être pris en considération.

3 Les mains dans le cambouis

3.1 Le marteau et l'enclume

Avant de commencer un développement avec XML, il faut se procurer un environnement de développement et des outils XML.

Le premier et le plus important de ces outils est l'éditeur XML. XML est du texte et peut donc être édité avec n'importe quel éditeur de texte. Sous UNIX, on peut par exemple utiliser `vi` ou `emacs`, sous Windows `notepad`. Il existe d'autres éditeurs qui sont orientés spécifiquement vers XML, mais la plupart sont payants.

Internet Explorer de Microsoft et Communicator de Netscape supportent tous deux XML et peuvent afficher des documents XML de façon hiérarchique (c'est-à-dire permettant un déplacement dans l'arbre). Comme la plupart des systèmes supportent au moins un de ces deux butineurs, il ne faut pas chercher très loin pour afficher un document XML. Les navigateurs Opera et Amaya du W3C prennent également XML en charge.

Notons ici que puisque l'un des premiers objectif d'XML est de décrire des données et non pas de les présenter, l'utilisation d'un navigateur n'est pas une nécessité. Comme XML est un standard ouvert, il peut être utilisé pour structurer des données facilement transferables d'un système à un autre. Vous n'avez donc pas besoin de vous contraindre à l'utilisation d'un navigateur Web pour valider vos données XML, de nombreux *parseurs* (analyseurs syntaxiques) vont faire le travail pour vous. Parmi ceux-ci, on peut citer Le parseur de James Clark, `expat`[10]. Sur le site d'Apache, on trouve également `Xerces`[11], un parseu en Java et C++.

3.2 Moteur !

Commençons avec un petit exemple[12] :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<film>
  <titre>X-Men</titre>
  <acteurs>
    <personne>Hugh Jackman</personne>
    <personne>Patrick Stewart</personne>
    <personne>Famke Janssen</personne>
  </acteurs>
```

[10] `http://www.jclark.com/xml/expat.html` ou `http://expat.sourceforge.net`
[11] `http://xml.apache.org`
[12] Disponible sur `http://www.cri.ensmp.fr/~silber/cours/xml/ex/xmen.xml`

```
  <realisateur>Bryan Singer</realisateur>
  <annee>2000</annee>
  <texte>Dans la lignée de <titre>Superman</titre></texte>
  <note>4</note>
  <producteur>Twentieth Century Fox</producteur>
</film>
```

Comme vous pouvez le voir, un document XML est un simple fichier texte. Celui-ci contient une critique de film, découpée en différentes sections. Chaque section est « marquée » avec des balises descriptives permettant d'identifier le type de donnée qu'elle contient.

Un document XML est soit ***bien formé*** soit ***valide*** (soit illégal mais ce n'est plus alors un document XML) :

bien formé se dit d'un document qui suit la recommendation XML, c'est-à-dire qui respecte les règles pour les noms d'***éléments*** et d'***attributs***, qui contient les ***déclarations*** essentielles et qui est parfaitement ***imbriqué*** ;

valide se dit d'un document bien-formé qui suit en plus les règles d'une définition de type de document (***DTD***, Document Type Definition) ou d'un ***Schéma XML***. En imposant une structure à un document XML, une DTD permet à un document de se conformer à certaines règles et évite donc des surprises aux applications utilisant ces données.

Les DTD sont essentielles lorsque l'on doit traiter un grand nombre de documents XML, puisqu'elles permettent d'appliquer un ensemble de règles standard pour différents documents et donc de respecter un certain standard. Cependant, pour des documents simples et petits, une DTD peut représenter un surcoût inutile en tant de traitement et de chargement.

3.3 Section par section

Tout document XML ***devrait*** commencer par une ***déclaration*** indiquant la version de XML utilisée. Cette déclaration est souvent appelée le ***prologue*** du document :

```
<?xml version="1.0"?>
```

Cette déclaration peut contenir des informations additionnelles, comme par exemple le type d'encodage du document et si le document doit être utilisé en combinaison avec une DTD externe ou d'autres entités. Par conséquence, le prologue d'un document XML ressemble souvent à ceci :

```
<?xml version="1.0" standalone="yes" ?>
```

ou à cela

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
```

Le prologue du document peut également contenir une ***déclaration de type de document***, pour spécifier des informations additionnelles sur celui-ci. Par exemple, l'emplacement de la DTD utilisée pour valider le document (si celle-ci est disponible), une liste optionnelle de déclarations d'***entités*** et ou le nom de l'élément ***racine*** du document.

Une déclaration de type de document ressemble souvent à ceci :

```
<!DOCTYPE racine DTD-externe
[
  ... déclarations diverses ...
]
>
```

Une déclaration de type de document possible pour la critique de film précédente pourrait ressembler à ceci :

```
<!DOCTYPE film SYSTEM "film.dtd">
```

Dans ce cas, le document devra être validé par rapport à la DTD `film.dtd` :

```
 <!ELEMENT film (titre,acteurs,
             realisateur,annee,
             texte,note?,producteur?)>
 <!ELEMENT titre (#PCDATA)>
 <!ELEMENT realisateur (#PCDATA)>
 <!ELEMENT annee (#PCDATA)>
 <!ELEMENT note (#PCDATA)>
 <!ELEMENT acteurs (personne+)>
 <!ELEMENT personne (#PCDATA)>
 <!ELEMENT producteur (#PCDATA)>
 <!ELEMENT texte ANY>
```

Si des déclarations d'entités sont présentes, on pourrait modifier la déclaration comme suit

```
<!DOCTYPE film SYSTEM "film.dtd"
[
  <!ENTITY Fox "Twentieth Century Fox">
  <!ENTITY Univ "Universal Pictures">
]
>
```

Nous parlerons des entités et de la DTD plus loin.

3.4 Élémentaire !

Le prologue du document est suivi d'une série d'***éléments***. Un élément est l'unité de base de XML, composé de ***données textuelles*** et de ***balises*** qui agrémentent celui-ci. Les frontières d'un élément sont définies par une balise de début et une balise de fin. Un élément peut également contenir des ***attributs*** descriptifs additionnels (de la forme `nom="valeur"`).

Voici trois exemples d'éléments XML sans attributs :

```
<titre>X-Men</titre>
<personne>Hugh Jackman</personne>
<année>2000</année>
```

XML permet également la définition d'***éléments vides***, c'est-à-dire qui n'ont pas de contenu et qui ne nécessitent pas de balise fermante. De tels éléments sont fermés en ajoutant une barre oblique (`/`) à la fin de leur balise ouvrante. Par exemple `<virgule/>` dans notre exemple est un élément vide.

Un élément doit obligatoirement commencer par une lettre, suivie éventuellement d'autres lettres et chiffres. Les noms d'éléments sont sensibles à la casse (minuscule et majuscule) et donc les éléments `<tOtO>` et `<ToTo>` sont deux éléments distincts.

Un élément peut ne contenir que du texte, comme

```
<réalisateur>Bryan Singer</réalisateur>
```

mais aussi une combinaison de texte et d'autres éléments :

```
<texte>Super<virgule/> dans la lignée
de <titre>Superman</titre></texte>
```

Pour être bien formé, un document XML doit contenir au moins un et un seul élément non vide le « plus externe ». Cet élément est appelé l'élément ***racine***. Cet élément peut à son tour contenir d'autres éléments, définissant ainsi un ***arbre***. Dans notre exemple, l'élément racine est `<film>...</film>`.

3.5 Données textuelles

Les contenus qui apparaîssent entre les balises ouvrantes et fermantes d'un document XML sont les données du document. Comme le précise la spécification XML « *tout texte qui n'est pas du balisage est une donnée textuelle du document* ». Bien que ces données puissent contenir des caractères alphanumériques ou des symboles, des précautions particulières doivent être prises quant aux caractères spéciaux comme `<`, `>` et `&`. Il faut les remplacer par les représentations hexadécimales correspondantes ou les chaînes `&lt;`, `&gt;` et `&amp;`.

3.6 Attributs

Les éléments peuvent contenir des attributs qui fournissent des informations additionnelles sur l'élément. Les attributs sont des paires (nom, valeur) de la forme `nom="valeur"` qui apparaîssent dans la balise ouvrante d'un élément. Par exemple, l'attribut `sexe` suivant permet de rajouter une donnée supplémentaire à l'élément `<personne>`.

```
<personne sexe="m">Hugh Jackman</personne>
<personne sexe="m">Patrick Stewart</personne>
<personne sexe="f">Famke Janssen</personne>
```

Les attributs doivent toujours apparaîtrent après le nom de l'élément et les noms d'attributs sont sensibles à la casse. Les valeurs d'attributs doivent toujours être entourés de guillemets (`"`) et le même attribut ne peut pas être répété plusieurs fois dans le même élément. Si votre document est lié à une DTD, il est possible d'édicter des règles sur les valeurs possibles d'un attribut.

Il est important de noter que la frontière entre un attribut et un élément est souvent très ténue, puisqu'ils ont une fonction similaire. Par exemple, alors qu'il est valide d'écrire

```
<personne sexe="m">Hugh Jackman</personne>
```

on peut obtenir exactement le même effet avec

```
<personne>
<name>Hugh Jackman</name>
<sexe>m</sexe>
</personne>
```

En d'autres termes, la décision d'utiliser un attribut ou un élément peut être un problème très compliqué et les besoins doivent être examinés au cas par cas. Les raisons valides pour utiliser des attributs au lieu d'éléments incluent l'affectation d'un identificateur à un élément spécifique

```
<critique num="124567">...</critique>
```

ou alors la description de caractéristiques de l'élément lui même

```
Le <animal couleur="rouge">loup</animal> saute au dessus de
l'<legume couleur="bleu">aubergine</legume>.
```

Si vous voulez restreindre les valeurs possibles d'un attributs, vous pouvez utiliser une DTD pour spécifier la liste des valeurs possibles et par défaut. Cette possibilité restreint les risques d'erreurs et de données incompatibles.

3.7 CDATA

Les blocs `CDATA` sont des sections de documents marquées explicitement comme ne contenant pas de marquage et qui sont donc traitées comme des données textuelles par le parseur. Ces blocs peuvent contenir à peu près n'importe quoi (chaînes de caractères, nombres, symboles, hiéroglyphes égyptiens, etc.) et seront ignorés par le parseur.

Un bloc `CDATA` typique commence par

```
<![CDATA[
```

et finit par

```
]]>
```

avec des données entre les deux.

Les blocs CDATA permettent d'ajouter des portions de texte importantes (comprenant des caractères spéciaux, des symboles ou du code de programme) à un document XML, pour que le parseur traîte ce texte comme des données régulières. Ainsi, alors qu'un parseur aura quelques problèmes avec ce document[13],

```
<?xml version="1.0" encoding="iso-8859-1"?>
<message_secret>
<de>notre homme a Paris</de>
<a>Directeur des operations speciales</a>
<texte_code>
12637 0%%348 83483 89238 82383 10341 0*049 27216 02039
3@492 83%84 22829 238#3 92345 72310 53467 12941 92461
46101 4<356 74(@1 4!128 47353 #511~ 47>~7 12942 38#53
</texte_code>
</message_secret>
```

celui-ci[14] passera sans problème

```
<?xml version="1.0" encoding="iso-8859-1"?>
<message_secret>
<de>notre homme a Paris</de>
<a>Directeur des operations speciales</a>
<texte_code>
<![CDATA[
12637 0%%348 83483 89238 82383 10341 0*049 27216 02039
3@492 83%84 22829 238#3 92345 72310 53467 12941 92461
46101 4<356 74(@1 4!128 47353 #511~ 47>~7 12942 38#53
]]>
</texte_code>
</message_secret>
```

Évidemment, il n'est pas possible d'inclure la séquence `]]>` dans un bloc `CDATA`. Si vous voulez inclure cette séquence dans un bloc `CDATA`, elle doit être écrite `]]>`.

[13] `http://www.cri.ensmp.fr/~silber/cours/xml/ex/messagebad.xml`
[14] `http://www.cri.ensmp.fr/~silber/cours/xml/ex/messageok.xml`

3.8 Instructions de commande

En plus des données textuelles, XML permet également l'inclusion d'instructions de commande à passer à l'application traitant le document. Ces instructions sont appelées ***instructions de commande***. Ces instructions ne font pas partie des données textuelles mais sont simplement passées telles quelles à l'application traitant le document. Celle-ci à le choix d'utiliser ces instructions ou de les ignorer.

Chaque instruction de commande inclut une cible (la chaîne de caractères utilisée pour identifier l'application pour laquelle l'instruction est destinée) et des données. Cette combinaison cible/données est entourée de balises `<?` et `?>`, comme dans l'exemple suivant :

```
<personne>Hugh Jackman<?photo grande?></personne>
<personne>Patrick Stewart<?photo grande?></personne>
<personne>Famke Janssen<?photo petite?></personne>
```

Notons toute première ligne d'un document XML est une instruction de commande précisant que c'est l'application `xml` qui doit être utilisée :

```
<?xml version="1.0"?>
```

Il existe également une commande `xml-stylesheet`[15] permettant de donner le nom d'un script de transformation ou d'une feuille de style pour le document XML :

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

3.9 Espace de noms

Un auteur de document est libre de choisir n'importe quel nom pour les éléments composant son document. Cette flexibilité peut poser un problème lorsque des éléments dans des documents différents ont le même nom. Par exemple, imaginons que l'on ait le document XML suivant[16] :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<revue>
 <titre>Cahiers du cinéma</titre>
 <numero>345</numero>
 <annee>2001</annee>
</revue>
```

Si on voulait combiner ce document avec notre document exemple, par exemple en ajoutant pour une revue de cinéma tous les films critiqués, il y aurait quelques problèmes, par exemple au niveau de la balise `<titre>` et de la balise `<annee>`.

C'est précisément pour lever ce genre d'ambiguïté que la spécification XML définit les ***espaces de noms***[17]. Ils permettent d'identifier de manière unique des éléments spécifiques dans un document XML. Pour cela, on ajoute un préfixe unique à un élément, l'associant ainsi à un ensemble de données.

La mise en place d'un espace de noms est très simple. La syntaxe est la suivante :

```
<nomElement xmlns:prefixe="URI">
```

Dans ce cas, le préfixe est la chaîne de caractères unique utilisée pour identifier l'espace de noms, qui est un URI. Voici l'exemple tiré de la documentation du W3C :

[15] `http://www.w3.org/TR/xml-stylesheet`
[16] `http://www.cri.ensmp.fr/~silber/cours/xml/ex/revue.xml`
[17] `http://www.w3.org/TR/REC-xml-names`

```
<x xmlns:edi='http://ecommerce.org/schema'>
  <!-- the "edi" prefix is bound to http://ecommerce.org/schema
       for the "x" element and contents -->
</x>
```

Un espace de noms est habituellement déclaré au niveau de l'élément racine, bien qu'un auteur de document XML soit libre de le déclarer à un plus bas niveau de la structure. Une fois que l'espace de noms est déclaré dans le document, il peut être utilisé en préfixant chaque élément dans cet espace de noms avec l'identificateur de l'espace de noms. Examinons par exemple notre document `<revue>` qui utilise un espace de noms `canards` pour éviter les *collisions* de noms.

```
<?xml version="1.0"  encoding="iso-8859-1"?>
<revue
 xmlns:canards="http://www.toto.com/ns/canards">
 <canards:titre>Cahiers du cinéma</canards:titre>
 <canards:numéro>345</canards:numéro>
 <canards:année>2001</canards:année>
</revue>
```

On peut également faire la même chose avec notre document exemple `<film>` en créant un espace de noms `critiques` :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<film
 xmlns:critiques="http://www.toto.com/ns/critiques">
 <critiques:titre>X-Men</critiques:titre>
 <critiques:acteurs>
  <critiques:personne>Hugh Jackman</critiques:personne>
  <critiques:personne>Patrick Stewart</critiques:personne>
  <critiques:personne>Famke Janssen</critiques:personne>
 </critiques:acteurs>
<critiques:realisateur>Bryan Singer</critiques:realisateur>
<critiques:annee>2000</critiques:annee>
<critiques:texte>
Dans la lignee de
<critiques:titre>Superman</critiques:titre>
</critiques:texte>
 <critiques:note>4</critiques:note>
</film>
```

L'URL de l'espace de noms est simplement un pointeur vers une adresse Web mais n'a pas de signification réelle et n'a même pas besoin d'être une adresse valide.

Si un document contient plusieurs espaces de noms, ajouter des déclarations d'espaces de noms et des préfixes à chaque élément peut devenir ardu. Un espace de noms par défaut peut donc être défini, qui permet d'omettre une partie des préfixes. On définit un espace de noms par défaut en ne mettant pas de préfixe dans la déclaration de l'espace des noms :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<film
 xmlns="http://www.toto.com/ns/critiques">
 <titre>X-Men</titre>
 <acteurs>
```

```
  <personne>Hugh Jackman</personne>
  <personne>Patrick Stewart</personne>
  <personne>Famke Janssen</personne>
 </acteurs>
<realisateur>Bryan Singer</realisateur>
<annee>2000</annee>
<texte>
Dans la lignee de
<titre>Superman</titre>
</texte>
 <note>4</note>
</film>
```

Ainsi, si on mélange nos deux exemples, il n'y aura pas de collision :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<film xmlns="http://www.toto.com/ns/critiques"
 xmlns:canards="http://www.toto.com/ns/canards">
 <titre>X-Men</titre>
 <acteurs>
  <personne>Hugh Jackman</personne>
  <personne>Patrick Stewart</personne>
  <personne>Famke Janssen</personne>
 </acteurs>
<realisateur>Bryan Singer</realisateur>
<annee>2000</annee>
<texte>
Dans la lignee de
<titre>Superman</titre>.
Les
<canards:titre>
Cahiers du cinéma
</canards:titre>
n'aiment pas ce film.
</texte>
 <note>4</note>
</film>
```

3.10 Commentaires

Comme en HTML et dans la plupart des langages de programmation, XML permet l'insertion de commentaires, dans un document. Les commentaires sont ignorés par le parseur et sont uniquement utiles dans un soucis de compréhension du document. Les commentaires peuvent apparaître n'importe où dans un document XML. Un commentaire à la forme suivante :

```
<!-- texte -->
```

4 Comment structurer les documents XML, la DTD

Une ***définition de type de document*** ou ***Document Type Definition*** (DTD) permet de définir un langage XML particulier. Une DTD permet de définir la ***grammaire*** et la ***syntaxe*** d'un langage XML particulier.

Un document XML bien formé peut être ***validé*** par rapport à une DTD. Si la validation réussit, le document est dit *valide par rapport à une DTD particulière*.

Voici par exemple la DTD de notre exemple récurrent (`film.dtd`[18]) :

```
<!ELEMENT film (titre,acteurs,
            realisateur,annee,
            texte,note?,producteur?)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT realisateur (#PCDATA)>
<!ELEMENT annee (#PCDATA)>
<!ELEMENT note (#PCDATA)>
<!ELEMENT acteurs (personne+)>
<!ELEMENT personne (#PCDATA)>
<!ELEMENT producteur (#PCDATA)>
<!ELEMENT texte ANY>
```

Une DTD peut être directement définie dans un document XML (dans sa déclaration de type de document) :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE film [
<!ELEMENT film (titre,acteurs,
            réalisateur,année,
            texte,note?,producteur?)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT réalisateur (#PCDATA)>
<!ELEMENT année (#PCDATA)>
<!ELEMENT note (#PCDATA)>
<!ELEMENT acteurs (personne+)>
<!ELEMENT personne (#PCDATA)>
<!ELEMENT texte ANY>
]>
<film>
  <titre>X-Men</titre>
  <acteurs>
    <personne>Hugh Jackman</personne>
    <personne>Patrick Stewart</personne>
    <personne>Famke Janssen</personne>
  </acteurs>
  <realisateur>Bryan Singer</realisateur>
  <annee>2000</annee>
  <texte>Dans la lignée de <titre>Superman</titre></texte>
  <note>4</note>
  <producteur>Twentieth Century Fox</producteur>
</film>
```

Une DTD peut également être définie de manière externe :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE film SYSTEM "film.dtd">
<film>
```

[18] `http://www.cri.ensmp.fr/~silber/cours/xml/ex/film.dtd`

```
  <titre>X-Men</titre>
  <acteurs>
    <personne>Hugh Jackman</personne>
    <personne>Patrick Stewart</personne>
    <personne>Famke Janssen</personne>
  </acteurs>
  <realisateur>Bryan Singer</realisateur>
  <annee>2000</annee>
  <texte>Dans la lignée de <titre>Superman</titre></texte>
  <note>4</note>
  <producteur>Twentieth Century Fox</producteur>
</film>
```

4.1 Déclaration d'un élément

La syntaxe de déclaration d'un élément est la suivante :

```
<!ELEMENT nom-d'élément (contenu-de-l'élément)>
```

`nom-d'élément` doit être un nom de balise valide (c'est-à-dire commençant par une lettre et suivi éventuellement de lettres et chiffres.
`contenu-de-l'élément` définit le ou les contenus valides pour un élément.

4.2 Contenus simples

Pour définir les contenus valides d'un élément, on peut utiliser les choses suivantes :
`#PCDATA` désigne des données textuelle qui seront traitées par le parseur.
`#CDATA` désigne des données textuelle qui ne seront pas traitées par le parseur.
`EMPTY` signifie que l'élément ne peut rien contenir.
`ANY` signifie que l'élément peut contenir n'importe quoi.

4.3 Contenus composés

Le contenu valide d'un élément peut être un autre élément, comme par exemple

```
<!ELEMENT acteurs (personne)>
```

qui spécifie que le contenu valide de l'élément `acteurs` est un élément `personne`.

Les caractères `+`, `*` et `?` permettent de spécifier que le contenu valide est respectivement *un ou plus* éléments

```
<!ELEMENT acteurs (personne+)>
```

aucun ou plus éléments

```
<!ELEMENT acteurs (personne*)>
```

ou *zéro ou un* élément

```
<!ELEMENT acteurs (personne?)>
```

D'autre part, il est possible de spécifier que le contenu valide est une séquence d'éléments :

```
<!ELEMENT film (titre,acteurs,réalisateur,année,
      texte,note)>
```

ou encore

```
<!ELEMENT film (titre+,#PCDATA,réalisateur*,année?,
      texte,note)>
```

Il est également possible de définir une disjonction d'éléments avec la barre oblique (et on peut même utiliser des parenthèses) :

```
<!ELEMENT film (titre,acteurs,(réalisateur|producteur),(année|mois),
      texte,note)>
```

Dans un document XML suivant cette DTD, les éléments doivent apparaître dans l'ordre où ils sont définis par la séquence.

4.4 Déclarations d'attributs

Les ***attributs*** possibles d'un élément sont déclarés avec une déclaration `ATTLIST` :

```
<!ATTLIST nom-d'élément nom-d'attribut type-d'attribut
          valeur-par-défaut>
```

Le `type-d'attribut` peut prendre les valeurs suivantes :
- `CDATA` pour des données textuelles,
- `(val|val|val|...)` un ensemble de valeurs,
- `ID` la valeur est un identificateur unique,
- `IDREF` la valeur est l'identificateur d'un autre élément,
- `IDREFS` la valeur est une liste d'identificateurs d'autres éléments,
- `NMTOKEN` la valeur est un nom XML valide (c'est-à-dire qu'il doit suivre les restrictions d'XML concernant les noms),
- `NMTOKENS` la valeur est une liste de noms XML valides,
- `ENTITY` la valeur est une entité,
- `ENTITIES` la valeur est une liste d'entités,
- `xml :`*`val`* la valeur est prédéfinie par XML (par exemple `xml :lang` ou encore `xml :space`).

La `valeur-par-défaut` peut prendre les valeurs suivantes :
- `value` l'attribut a une valeur par défaut,
- `#REQUIRED` La valeur de l'attribut doit être incluse dans l'élément,
- `#IMPLIED` La valeur de l'attribut n'est pas nécessaire,
- `#FIXED value` La valeur de l'attribut doit exister et être égale à celle spécifiée.

4.5 Exemples de déclarations d'attributs

```
<!ELEMENT carré (EMPTY)>
<!ATTLIST carré taille CDATA "0">
...
<carré taille="100"></carré>
```

Dans l'exemple précédent, l'élément `carré` est défini comme un élément vide avec un attribut `taille` qui a une valeur par défaut `"0"`.

4.6 Valeur d'attribut par défaut

Spécifier une valeur par défaut pour un attribut assure que l'attribut à une valeur même si l'auteur du document ne lui donne pas de valeur.
Syntaxe :

```
<!ATTLIST nom-d'élément nom-d'attribut type-d'attribut "valeur">
```

Exemple de DTD :

```
<!ATTLIST paiement type CDATA "chèque">
```

Exemple XML :

```
<paiement type="CB">...</paiement>
<paiement>...</paiement>
```

4.7 Valeur optionnelle (`IMPLIED`)

Une valeur optionnelle permet de ne pas obliger l'auteur à donner une valeur pour un attribut.
Syntaxe :

```
<!ATTLIST nom-d'élément nom-d'attribut type-d'attribut #IMPLIED>
```

Exemple de DTD :

```
<!ATTLIST contact fax CDATA #IMPLIED>
```

Exemple XML :

```
<contact fax="0164694847">...</contact>
<contact>...</contact>
```

4.8 Valeur obligatoire (`REQUIRED`)

Une valeur obligatoire oblige l'auteur à donner une valeur à un attribut.
Syntaxe :

```
<!ATTLIST nom-d'élément nom-d'attribut type-d'attribut #REQUIRED>
```

Exemple de DTD :

```
<!ATTLIST contact fax CDATA #REQUIRED>
```

Exemple XML :

```
<contact fax="0164694847">...</contact>
```

4.9 Valeur fixe (`FIXED`)

Une valeur fixe indique une valeur obligatoire et constante pour un attribut.
Syntaxe :

```
<!ATTLIST nom-d'élément nom-d'attribut type-d'attribut #FIXED "valeur">
```

Exemple de DTD :

```
<!ATTLIST windows compagnie CDATA #FIXED "Microsoft">
```

Exemple XML :

```
<windows compagnie="Microsoft">...</windows>
```

4.10 Attributs énumérés

Un attribut de type énuméré est utilisé pour définir un ensemble de valeurs possible pour l'attribut. Une valeur différente provoquera une erreur.
Syntaxe :

```
<!ATTLIST nom-d'élément nom-d'attribut (val|val|...) valeur-par-défaut>
```

Exemple de DTD :

```
<!ATTLIST paiement type (CB|chèque) "chèque">
```

Exemple XML :

```
<paiement type="chèque">
<paiement type="CB">
```

4.11 Entités

Les ***entités*** permettent de référencer quelque chose, soit un nom de variable utilisable dans le document XML (entité générale), soit un alias utilisable uniquement dans la DTD (entité paramètre).

Entité générale Une entité générale interne (variable) est un nom référençant un emplacement mémoire contenant du texte. Une fois qu'une entité interne a été définie, les auteurs de documents XML peuvent utiliser ce nom à la place du contenu de la mémoire correspondante.

Les entités générales internes sont particulièrement utiles lorsqu'un morceau de texte se répète à plusieurs endroits dans le document (un nom, une adresse mail, un entête ou un pied de page, etc.). En définissant une entité pour contenir des données récurrentes, XML permet aux auteurs de documents d'effectuer un changement global en modifiant une seule valeur.

Ainsi, dans l'exemple suivant[19], on peut utiliser l'entité `Fox` pour désigner la `Twentieth Century Fox` :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE film SYSTEM "film.dtd"
[
  <!ENTITY Fox "Twentieth Century Fox">
]
>
<film>
  <titre>X-Men</titre>
  <acteurs>
    <personne>Hugh Jackman</personne>
    <personne>Patrick Stewart</personne>
    <personne>Famke Janssen</personne>
  </acteurs>
  <realisateur>Bryan Singer</realisateur>
  <annee>2000</annee>
  <texte>Dans la lignée de <titre>Superman</titre></texte>
  <note>4</note>
  <producteur>&Fox;</producteur>
</film>
```

Les entités doivent être déclarées avant d'être utilisées et doivent apparaître dans la déclaration de type de document. XML contient cinq entités générales internes prédéfinies :

[19] `http://www.cri.ensmp.fr/~silber/cours/xml/ex/xmendtdent.xml`

– `<` pour le symbole <
– `>` pour le symbole >
– `&apos;` pour le symbole '
– `&quote;` pour le symbole "
– `&` pour le symbole &

Les entités générales internes peuvent contenir des balises XML et une entité peut contenir des références à d'autres entités. Par contre, une entité ne peut pas se référencer elle-même, puisque l'on obtient dans ce cas une définition récursive. Par exemple, il est interdit de définir :

```
<!ENTITY GNU "&GNU; is not UNIX">
```

Entité paramètre Ce type d'entité permet la déclaration d'un paramètre utilisable uniquement dans la DTD. La DTD doit être externe au document pour pouvoir utiliser des entités paramètre (PE en anglais et dans la documentation XML). Voici par exemple un fichier `equipe.xml`[20] :

```
<?xml version="1.0"?>
<!DOCTYPE equipe SYSTEM "equipe.dtd">
<equipe>
<joueur taille="cm">180</joueur>
</equipe>
```

et sa DTD `equipe.dtd`[21] associée :

```
<!ENTITY % taille.att 'taille CDATA #REQUIRED'>
<!ELEMENT equipe (joueur)+>
<!ELEMENT joueur (#PCDATA)>
<!ATTLIST joueur %taille.att;>
```

Une entité paramètre se déclare et s'utilise avec le signe %.

5 Les schémas XML

Les schémas XML permettent, comme les DTD, de représenter la grammaire d'une classe de documents XML. À la différence des DTD, les schémas permettent de typer les données se trouvant dans le document (à la différence d'une DTD où il y a un seul type de données, `CDATA`). Une autre différence par rapport à une DTD est qu'un schéma est exprimé en XML, ce qui permet à une grammaire d'être manipulée comme n'importe quel document XML. D'autre part, un schéma permet de préciser le nombre d'occurrences d'un élément au sein d'un document XML plus précisément qu'avec une DTD. Finalement, nous allons voir qu'un schéma offre plus de liberté quant à l'expression de l'imbrication des éléments au sein d'un document XML.

Revenons à notre exemple de film pour voir ce que donnerait notre DTD transformée en schéma XML[22] :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsd:schema
  xmlns:xsd="http:/www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
<xsd:element name="film" type="typeFilm"/>
<xsd:complexType name="typeFilm">
  <xsd:sequence>
```

[20] `http://www.cri.ensmp.fr/~silber/cours/xml/ex/equipe.xml`
[21] `http://www.cri.ensmp.fr/~silber/cours/xml/ex/equipe.dtd`
[22] `http://www.cri.ensmp.fr/~silber/cours/xml/ex/film.xsd`

```
    <xsd:element name="titre" type="xsd:string"/>
    <xsd:element name="acteurs" type="typeActeur"/>
    <xsd:element name="realisateur" type="xsd:string"/>
    <xsd:element name="annee" type="xsd:decimal"/>
    <xsd:element name="texte" type="xsd:string"/>
    <xsd:element name="note" type="xsd:decimal"
      minOccurs="0" maxOccurs="1"/>
    <xsd:element name="producteur" type="xsd:string"
      minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="typeActeur">
  <xsd:sequence>
    <xsd:element name="personne" type="xsd:string"
         minOccurs="1" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Notre document XML utilisant ce schéma est le suivant[23] :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<film
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="film.xsd">
  <titre>X-Men</titre>
  <acteurs>
    <personne>Hugh Jackman</personne>
    <personne>Patrick Stewart</personne>
    <personne>Famke Janssen</personne>
  </acteurs>
  <realisateur>Bryan Singer</realisateur>
  <annee>2000</annee>
  <texte>Dans la lignée de <titre>Superman</titre></texte>
  <note>4</note>
  <producteur>Twentieth Century Fox</producteur>
</film>
```

Définition des éléments L'ajout d'un élément utilise la syntaxe suivante :

```
<xsd:element name="titre" type="xsd:string"/>
```

Le type `xsd :string` est un type de base défini par XML Schéma. D'autres types de base existent, comme `xsd :boolean`, `xsd :float`, `xsd :decimal`, `xsd :string`, `xsd :date`, etc. Il est possible de définir un nouveau type en utilisant un nouveau nom :

```
<xsd:element name="film" type="typeFilm"/>
```

Il est ensuite nécessaire de définir le contenu du nouveau type, comme nous l'avons fait dans l'exemple précédent :

[23] `http://www.cri.ensmp.fr/~silber/cours/xml/ex/xmenschema.xml`

```
 <xsd:complexType name="typeFilm">
  <xsd:sequence>
    ...
  </xsd:sequence>
</xsd:complexType>
```

Ici nous avons défini un type qui est une séquence contenant une liste ordonnée de différents éléments devant apparaître dans cet ordre. Il existe d'autres constructions disponibles utilisables à la place d'une séquence : `<xsd :choice>` qui permet de lister une disjonction d'éléments, `<xsd :all>` qui permet de faire apparaître les sous-éléments une fois ou pas du tout, et dans n'importe quel ordre.

Il est possible de donner le nombre d'occurrences possibles d'un élément avec les attributs `minOccurs` et `maxOccurs` :

```
<xsd:element name="note" type="xsd:decimal"
  minOccurs="0" maxOccurs="1"/>
```

Définition des attributs Il est possible de définir un ou plusieurs attributs dans un nouveau type, à l'aide de la construction `<xsd :attribute>`. Par exemple, pour rajouter un attribut à l'élément `film`, on procède de la façon suivante :

```
 <xsd:complexType name="typeFilm">
  <xsd:sequence>
    ...
  </xsd:sequence>
  <xsd:attribute name="genre" type="xsd:string"/>
</xsd:complexType>
```

De la même façon que pour une DTD, il est possible d'indiquer des contraintes d'occurrences sur un attribut, avec les attributs `use` et `value`. Par exemple :

```
<xsd:attribute name="genre" type="xsd:string"
   use="optional"/>
```

ou encore

```
<xsd:attribute name="genre" type="xsd:string"
   use="required" value="science-fiction"/>
```

Pour plus d'informations sur les schémas, se référer à la documentation officielle sur W3C.

6 Trouver son chemin avec XPath

Xpath permet de spécifier un ensemble de ***chemins*** à travers un document et donc de désigner un sous ensemble du document.

Ainsi, notre exemple de la critique de film aura la représentation en arbre suivante :

```
<film>
  |--<titre>
       CDATA: X-Men
  |--<acteurs>
         |--<personne>
              CDATA: Hugh Jackman
         |--<personne>
              CDATA: Patrick Stewart
         |--<personne>
              CDATA: Famke Janssen
     ...
```

L'élément `titre` est caractérisé par le chemin :

```
/film/titre
```

Les différents nœuds sont séparés par une barre oblique. Un certain nombre de mots clés permettent de spécifier des chemins compliqués (`self`, `parent`, `child`, `attribute`, `ancestor`, `descendant`, etc.) en utilisant `::` devant le nom d'un nœud. Par exemple, le chemin précédent est équivalent à :

```
/child::film/child::titre
```

sachant que `child ::` est sous-entendu quand rien n'est précisé.
Les fonctions prédéfinies `text()`, `comment()`, `processing-instruction()` et `node()` permettent de caractériser respectivement les données textuelles, les commentaires, les instructions et finalement tout. Par exemple, le texte du titre est caractérisé par :

```
/film/titre/text()
```

6.1 Autres exemples

Tous les fils de `film` :

```
/film/node()
```

ou

```
/film/*
```

Il est possible d'accéder à un attribut `toto` avec la notation `attribute ::toto` ou la notation abrégée `@toto`. Pour accéder à tous les attributs fils de `film`

```
/film/@*
```

Tous les attributs `sexe` de `personne`

```
/film/acteurs/personne/@sexe
/film/acteurs/personne/attribute::sexe
```

Il est possible de spécifier un prédicat entre crochets après une expression XPath. Par exemple, pour accéder au second rôle

```
/film/acteurs/node()[2]
```

Le premier élément sous la racine (ici `film`), abbréviation de `/node()` :

```
/*
```

Tous les éléments du document (abbréviation de `/descendant-or-self ::*`) :

```
//*
```

Pour plus d'informations sur XPath, se référer à la documentation officielle : `http://www.w3.org/TR/xpath`.

7 Du style avec XSLT

Maintenant que nous savons comment mettre en forme nos données avec XML, nous allons voir comment utiliser ces données : mise en forme, extraction, etc.

Une des possibilités est d'utiliser XSLT, Extensible Stylesheet Language Transformations. Nous allons voir comment XSLT fonctionne et quelques exemples d'utilisation.

7.1 XSL et XSLT ?

XSL, Extensible Language Stylesheet[24], est un langage pour définir la présentation et la mise en forme des données XML. Alors qu'XSL a été initialement développé comme un langage unique, il a été par la suite divisé en deux composantes indépendantes :

- un langage pour *transformer* les documents XML en réorganisant et restructurant les arbres XML, **XSLT**[25],
- un vocabulaire pour prendre en charge la mise en forme et la présentation du résultat (appelé **XSL-FO**).

Nous allons surtout examiner XSLT qui permet de transformer un document XML en à peu près n'importe quoi.

7.2 Arbre XML

Une transformation avec XSLT consiste essentiellement à transformer un arbre XML en un nouvel arbre. Cette transformation s'effectue avec une ***feuille de style*** XSLT qui contient une ou plusieurs règles de transformation.
Une règle de transformation effectue deux opérations :

- identifier un ***motif*** dans l'arbre,
- définir une nouvelle structure d'arbre résultante.

Considérons l'exemple suivant :

```
<xsl:template match="film">
  Le titre du film est <xsl:value-of select="titre" />
</xsl:template>
```

Dans ce cas, le motif (`template`) est la ***racine*** (`film`) et on recherche l'élément `titre`. L'opération effectuée est l'affichage du titre du film.
Le corps du motif peut contenir des caractères affichés tels quels et/ou des instructions XSLT qui seront exécutées. Par exemple, le règle de motif suivante contient une instruction XSLT qui boucle un certain nombre de fois :

```
<xsl:template match="/acteurs">
  <xsl:for-each select="personne">
  Acteur: <xsl:value-of select="."/><br/>
  </xsl:for-each>
</xsl:template>
```

Une feuille de style peut contenir plusieurs règles de motifs. Chaque règle s'applique à un élément spécifique (ou à un ensemble d'éléments) de l'arbre XML, exécute un ensemble d'instructions pour créer la structure des nœuds résultants et spécifie si le traitement doit continuer récursivement (vers les ***enfants*** de l'élément courant) ou s'arrêter à cet endroit.
Si le traitement doit continuer récursivement, le processeur XSLT recherche une règle de motif correspondant à l'élément suivant dans l'arbre source, crée le fragment d'arbre résultant et continue récursivement jusqu'à ce qu'il n'y ait plus d'éléments à traiter. Le résultat final est donc produit par l'interaction entre les différentes règles de motif.
Au cas où plusieurs règles correspondent à un élément de l'arbre source, XSLT tente de résoudre le conflit en appliquant la règle la plus spécifique.

[24] `http://www.w3.org/TR/xsl`
[25] `http://www.w3.org/TR/xslt`

7.3 XML+XSLT=HTML

Reprenons notre exemple de critique de film[26] :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<film>
  <titre>X-Men</titre>
  <acteurs>
    <personne>Hugh Jackman</personne>
    <personne>Patrick Stewart</personne>
    <personne>Famke Janssen</personne>
  </acteurs>
  <realisateur>Bryan Singer</realisateur>
  <annee>2000</annee>
  <texte>Dans la lignée de <titre>Superman</titre></texte>
  <note>4</note>
  <producteur>Twentieth Century Fox</producteur>
</film>
```

que nous voudrions transformer en un document HTML[27] :

```
<html>
<head>
<META http-equiv="Content-Type"
  content="text/html; charset=UTF-8">
Le film du jour</head>
<body>
<h1>X-Men</h1>
<h3>Un film de Bryan Singer
      (2000)</h3>
<p>
    Hugh Jackman,

    Patrick Stewart

   et Famke Janssen.

  </p>
  Note: 4</body>
</html>
```

Voici la feuille de style qui effectue la transformation[28] :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="film">
  <html>
  <head>Le film du jour</head>
  <body>
```

[26] http://www.cri.ensmp.fr/~silber/cours/xml/ex/xmen.xml
[27] http://www.cri.ensmp.fr/~silber/cours/xml/ex/xmen.html
[28] http://www.cri.ensmp.fr/~silber/cours/xml/ex/filmdujour.xsl

```
  <xsl:apply-templates select="titre"/>
  <h3>Un film de <xsl:value-of select="realisateur"/>
      (<xsl:value-of select="annee"/>)</h3>
  <p><xsl:apply-templates select="acteurs"/></p>
  Note: <xsl:value-of select="note"/></body></html>
 </xsl:template>
 <xsl:template match="titre">
  <h1><xsl:value-of select="."/></h1>
 </xsl:template>
 <xsl:template match="acteurs">
  <xsl:apply-templates/>
 </xsl:template>
 <xsl:template match="personne[position() != last()]">
  <xsl:value-of select="." />,
 </xsl:template>
 <xsl:template match="personne[position() = (last()-1)]">
  <xsl:value-of select="." />
 </xsl:template>
 <xsl:template match="personne[position() = last()]">
   et <xsl:value-of select="." />.
 </xsl:template>
</xsl:stylesheet>
```

Les fonctions `position()` et `last()` sont des fonctions XPath.

7.4 Un peu plus de style

Dans l'exemple précédent, nous n'avons pas affiché le `texte` de la critique. Nous voudrions maintenant obtenir le document HTML suivant[29] :

```
<html>
<head>
<META http-equiv="Content-Type"
content="text/html; charset=UTF-8">Le film du jour</head>
<body>
<h1>X-Men</h1>
<h3>Un film de Bryan Singer
      (2000)</h3>
<p>
    Hugh Jackman,

    Patrick Stewart

   et Famke Janssen.

  </p>
<p>Super dans la lign&eacute;e de <i>Superman</i>
</p>
  Note: 4</body>
</html>
```

[29] `http://www.cri.ensmp.fr/~silber/cours/xml/ex/xmen2.html`

Voici un extrait de la feuille de style XSL modifiée[30] :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

 <xsl:template match="/film">
  <html>
  <head>Le film du jour</head>
  <body>
  <xsl:apply-templates select="titre"/>
  <h3>Un film de <xsl:value-of select="réalisateur"/>
      (<xsl:value-of select="année"/>)</h3>
  <p><xsl:apply-templates select="acteurs"/></p>
  <p><xsl:apply-templates select="texte"/></p>
  Note: <xsl:value-of select="note"/></body></html>
 </xsl:template>

 <xsl:template match="titre">
  <h1><xsl:value-of select="."/></h1>
 </xsl:template>

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

 <xsl:template match="texte//titre">
  <i><xsl:value-of select="." /></i>
 </xsl:template>


 <xsl:template match="acteurs">
  <xsl:apply-templates/>
 </xsl:template>
 <xsl:template match="personne[position() != last()]">
  <xsl:value-of select="." />,
 </xsl:template>
 <xsl:template match="personne[position() = (last()-1)]">
  <xsl:value-of select="." />
 </xsl:template>
 <xsl:template match="personne[position() = last()]">
   et <xsl:value-of select="." />.
 </xsl:template>
</xsl:stylesheet>
```

On peut remarquer que le motif rajouté (`texte//titre`) permet d'identifier un titre de film dans le texte de la critique et de l'afficher différemment du titre du film critiqué.

8 Liens

cours `http://www.cri.ensmp.fr/~silber/cours/xml`

[30] `http://www.cri.ensmp.fr/~silber/cours/xml/ex/filmdujour2.xsl`

expat http://www.jclark.com/xml/expat.html

xml-apache http://xml.apache.org

w3c http://www.w3.org

$Id: poly.tex,v 1.22 2003/01/09 10:35:47 silber Exp $