La technologie XML

Bernd Amann

amann@cnam.fr

Cnam

Objectifs du Cours

Mieux comprendre l'utilisation de la technologie XML pour la gestion des données sur le Web :

Représentation de données avec XML

Interrogation avec XPath

Transformation avec XSLT

Stockage de documents/données XML

Introduction à XML

Pourquoi XML?

Une présentation, basée sur une étude de cas, des apports de XML :

- XML, format universel
- Publication avec XSLT
- Échange et intégration de données
- XML et bases de données

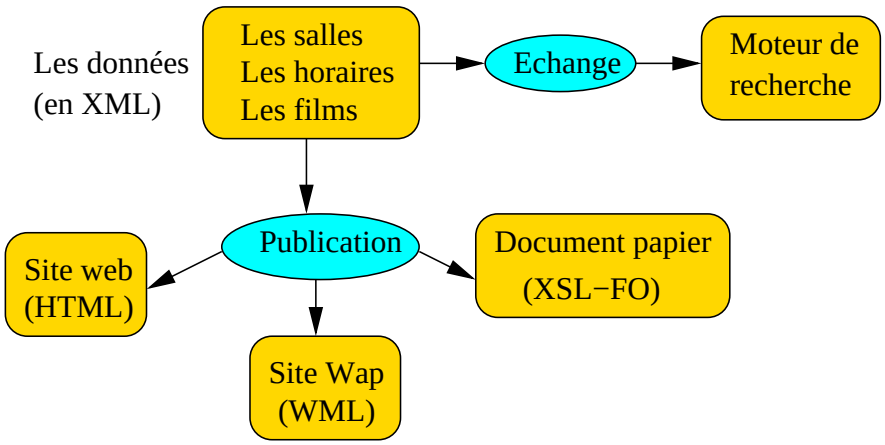
Les questions : XML, c'est quoi, et pour quoi faire?

Une étude de cas

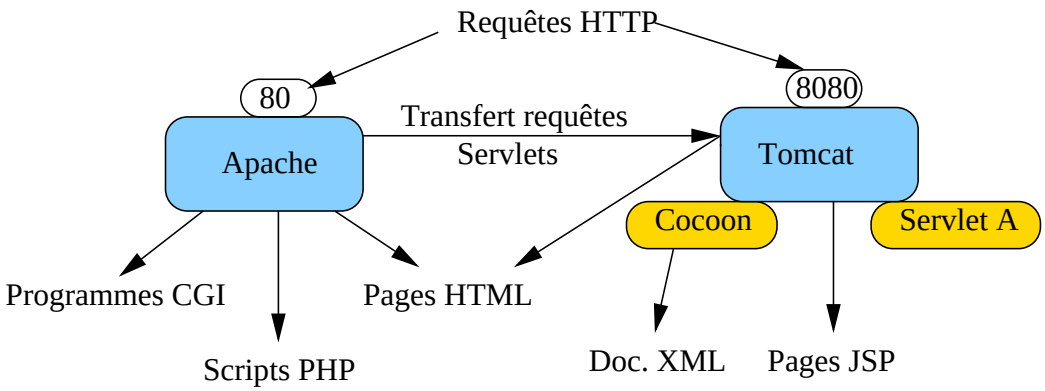
L'Officiel des spectacles !
On a des cinémas qui veulent

- *diffuser* leur programme (salles, séances) sur le Web, sur le WAP, sur des tracts et des affiches...
- *inclure* des informations sur des films provenant d'une base de données MySQL,
- *approvisionner* un moteur de recherche pour chercher des séances, des films, des horaires...

Echanges et publication des données



L'architecture du site



La fiche du film Gladiator

La fiche du film peut être publiée

- en HTML pour Netscape/IE
- en WML pour le portables WAP
- en SMIL pour Realplayer
- dans un moteur de recherche SallesEnLigne.com

L'information ?

- c'est la *même*, sous des formes différentes
- elle est échangée entre plusieurs acteurs

Les solutions XML ?

- **Format universel** :
  - ▷ représentation en chaîne de caractères d'un contenu structuré
  - ▷ indépendant de toute application
- **Publier l'information**
  - ▷ outils de transformation simples pour convertir un contenu XML
- **Échanger et intégrer l'information**
  - ▷ assembler des contenus XML, ou au contraire en extraire des informations

Pourquoi pas HTML ?

Actuellement, le principal format du Web est HTML:

- HTML est un langage pour **présenter** des informations à **l'écran**.
  - ▷ il ne permet pas d'échanger des données
  - ▷ il ne permet pas un traitement des données autre que l'affichage

On ne sait pas interpréter des données fournies en HTML.

XML, c'est quoi ?

XML = rendre un **contenu** accessible à toute application.

Le contenu :

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

Le même, en XML :

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

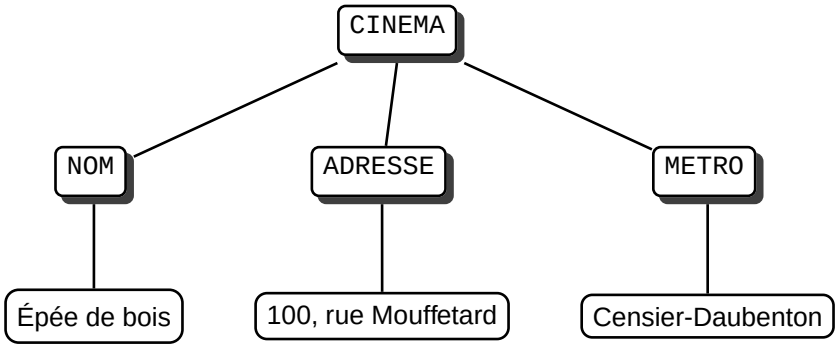

Le même, mieux présenté

Présentation courante : avec indentation

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

NB : il y a des espaces et des sauts de ligne

Encore mieux : sous forme d'arbre

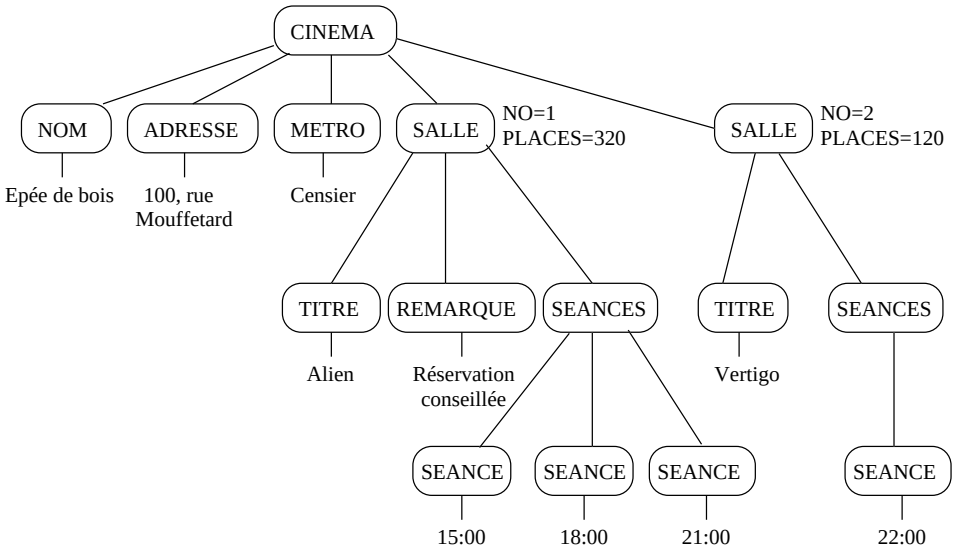


Traiter un document XML = extraire des informations d'un arbre.

Un exemple plus complet, avec attributs

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

  <SALLE NO='1' PLACES='320'>
    <TITRE>Alien</TITRE>
    <REMARQUE>Reservation conseillée</REMARQUE>
    <SEANCES>
       <SEANCE>15:00</SEANCE>
       <SEANCE>18:00</SEANCE>
       <SEANCE>21:00</SEANCE>
    </SEANCES>
 </SALLE>
</CINEMA>
```

Sous forme d'arbre



Documents XML

Un **document XML** est

- une **représentation alphanumérique**
- de **données (semi-)structurées**
- provenant
  - ▷ **d'un ou de plusieurs fichiers** et/ou
  - ▷ **d'un message** et/ou
  - ▷ **d'une base de données**...

Origine d'un document XML

XML permet **d'intégrer** des contenus provenant d'origines diverses

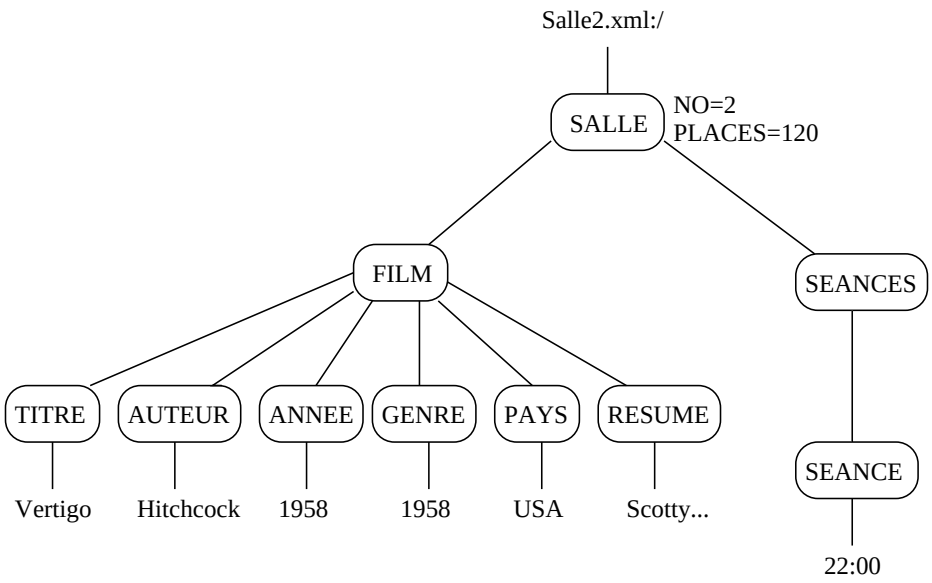
- d'un éditeur de texte
- d'un site Web
- d'une base de données
- d'un fichier...

Encore la même idée : rendre le contenu indépendant de l'application

Exemple d'Intégration

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<SALLE NO='2' PLACES='120'>
 <FILM>
  <TITRE>Vertigo</TITRE>
  <AUTEUR>Alfred Hitchcock</AUTEUR>
  <ANNEE>1958</ANNEE>
  <GENRE>Drame</GENRE>
  <PAYS>Etats Unis</PAYS>
  <RESUME>Scottie Ferguson, ancien inspecteur
       de police, est sujet au vertige depuis
       qu'il a vu mourir son collègue....
  </RESUME>
 </FILM>
 <SEANCES>
    <SEANCE>22:00</SEANCE>
 </SEANCES>
</SALLE>
```

Sous forme d'arbre



Le cinéma : intégration des salles

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE CINEMA [
  <!ENTITY salle1 SYSTEM "Salle1.xml">
  <!ENTITY salle2 SYSTEM "Salle2.xml">
]>
<CINEMA>
  <NOM>Epée de bois</NOM>
  <ADRESSE>100, rue Mouffetard</ADRESSE>
  <METRO>Censier-Daubenton</METRO>
  &salle1;
  &salle2;
</CINEMA>
```

Premier bilan

Disposer d'une et une seule représentation

- J'utilise un traitement de texte ? je suis prisonnier du format
- Je stocke dans une base de données ? idem

Créer un langage pour décrire nos données

- Ne pas utiliser HTML, dédié à la présentation dans un navigateur
- => utiliser XML, et convertir vers HTML

Publication de données avec XML

Séparer la gestion du contenu de la présentation

- Gestion du contenu => décrire nos informations, avec un vocabulaire XML
- Présentation => mettre en forme nos documents pour une application particulière

Application

Nous avons décrit notre cinéma avec notre propre langage (DTD). On peut traduire ce langage vers d'autres langages XML :

- HTML pour la présentation Web standard
- WML pour la présentation WAP
- SMIL pour une présentation multimédia
- XSL-FO pour la production de documents papier

Version HTML

HTML revisité :

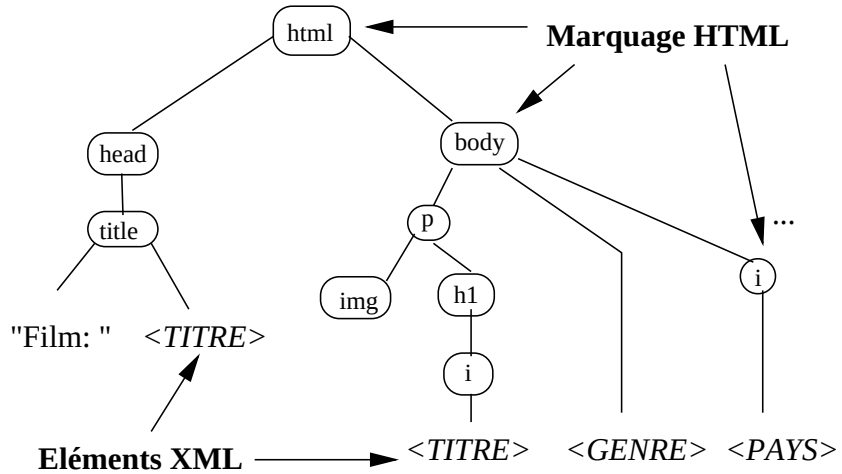
- Un document HTML **est** un document XML
- Le vocabulaire est fixé, ainsi que la syntaxe
- Chaque balise a une signification bien définie

HTML a été normalisé comme « dialecte » XML
=> XHTML

Ce qu'on veut obtenir (démo)

```
<html>
 <head><title>Film: Vertigo</title></head>
 <body bgcolor="white">
    <p><img SRC="Vertigo.png">
       <h1><i>Vertigo</i></h1>
       Drame, <i>Etats Unis</i>, 1958
    </p>
    <p>
    Mis en scène par  <b>Alfred Hitchcock</b>
    <h3>Résumé</h3>Scottie Ferguson,
     ancien inspecteur de police, est sujet
     au vertige depuis qu'il a vu mourir son
            collègue...
    </p>
 </body>
</html>
```


HTML, sous forme d'arbre



Transformation XSLT

Un programme de transformation XSLT doit nous permettre :

- De prendre en entrée un **document XML source**
- De produire en sortie un autre arbre XML
- D'insérer dans le document en sortie des fragments du document source

WML, autre dialecte de XML

- Document WML : marqué par la balise <wml>
- Il est divisé en *cartes*, unité d'affichage sur le mobile (<card>)
- Elements principaux :
  - ▷ des balises simples de mise en forme (<b>, <i>)
  - ▷ des *ancres* pour passer d'une carte à une autre

Exemple d'une carte WML

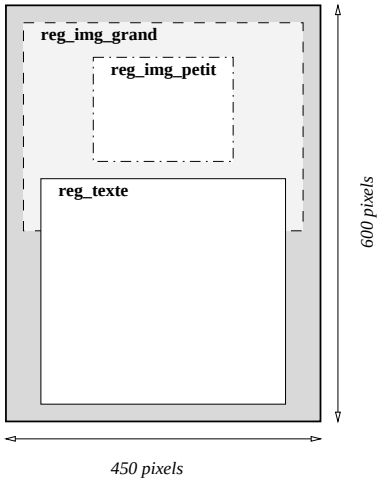
```
<?xml version="1.0"
      encoding="iso-8859-1"?>
<wml>
  <card>
    <p>
      <b>
        Alien
      </b>
      , 1979, Ridley Scott
    <br/>
    Près d&apos;un vaisseau spatial
    échoué sur une lointaine
    planète, ..
   </p>
  </card>
</wml>
```


SMIL

Langage pour la création de documents **multimédia** avec XML.

- On indique une fenêtre d'affichage avec différentes régions pour l'affichage des composants.
- On place les composants dans les différentes régions (positionnement spatiale).
- On synchronise l'affichage des composants (positionnement temporel).

Entête SMIL: Fenêtre et régions




```
<head>
 <meta  name="title"
        content="Version SMIL" />
 <meta  name="author"
        content="©2001 Sallesenligne.com" />
 <layout>
  <root-layout width="450" height="600" />
  <region id="reg_img_grand"
          width="400" height="300"
          left="25"  top="25" fit="meet" />
  <region id="reg_img_petit"
          width="200" height="125"
          left="125" top="75" fit="meet" />
  <region id="reg_texte"
          width="350" height="325"
          left="50"  top="250" z-index="1"/>
 </layout>
</head>
```


Corps SMIL: Positionnement


```
<body>
 <seq>
    <par endsync="first" id="page1" >
      <audio src="Sound.wav" />
      <a href="#page2" >
        <img src="Vertigo.png"
             region="reg_img_grand" />
      </a>
    </par>
    <par id="page2" dur="30s" >
      <text  src="Vertigo-Resume.txt"
             region="reg_texte" />
      <img   src="Vertigo.png"
             region="reg_img_petit" />
    </par>
  </seq>
</body>
```

Démo !

XSL-FO

Langage de **description** de documents avec XML.

- On indique les paramètres de mise en page (marges, taille des polices...)
- On place le contenu entre des balises de formatage

=> un processeur se charge de produire le document

Exemples d'un document XSL-FO

```
<?xml version="1.0" encoding="iso-8859-1"?>
<fo:root>
  <fo:layout-master-set>
    <fo:simple-page-master master-name="page"
      page-height="29.7cm" page-width="21cm"/>
  </fo:layout-master-set>
  <fo:page-sequence master-name='simple'>
    <fo:flow font-size="20pt">
       <fo:block>
         Ceci est le premier paragraphe,
       </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

Démo ! Le programme de l'Epée de Bois!

Quelques idées

J'ai **mes** données

- Je leur ai défini une représentation
- Je leur applique des traitements (publication ou autre)
- **Je peux les transmettre** à quelqu'un d'autre (tout ou partie)

=> un **service** externe m'apporte une valeur ajoutée

Quel format ?

Mon problème :

- J'ai décrit mes données avec **mon** langage XML
- L'application attend des données dans **son** langage

Il faut :

- Décrire formellement les deux langages
- Faire une **traduction** de l'un à l'autre

Les DTD

Document Type Definition

- Pour définir la **structure** d'une classe de documents (d'un langage)
- Exemple : un élément de type texte :
  `<!ELEMENT TITRE (#PCDATA) >`
- Exemple : un élément constitué d'un titre, d'un cinéma, d'une ville, d'une URL (optionnelle) et d'une séquence d'heures.

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

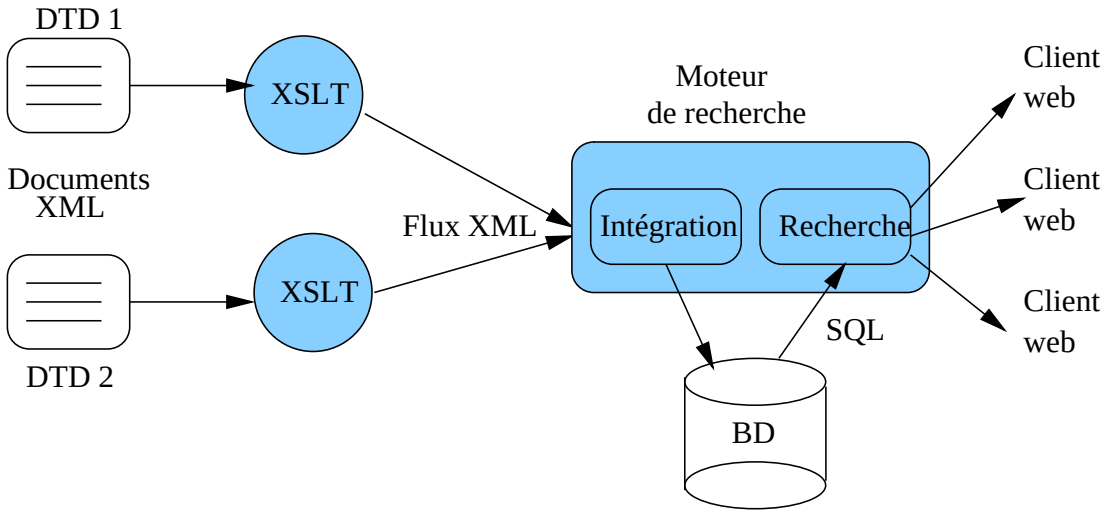

La DTD du moteur de recherche

Un fichier auquel on peut faire référence dans un document :

```
1  <!ELEMENT FILM    ( TITRE, CINEMA, VILLE, URL?, HEURE+ ) >
2  <!ELEMENT TITRE   ( #PCDATA ) >
3  <!ELEMENT CINEMA  ( #PCDATA ) >
4  <!ELEMENT VILLE   ( #PCDATA ) >
5  <!ELEMENT URL     ( #PCDATA ) >
6  <!ELEMENT HEURE   ( #PCDATA ) >
```

Document valide : conforme à une DTD.

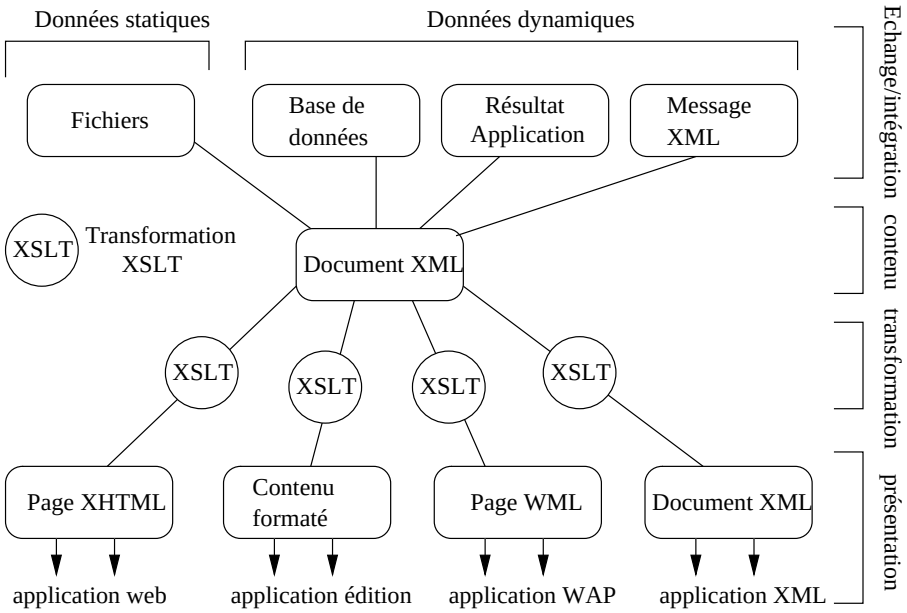
Architecture (Démo)



Le document intégrateur

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE MOTEUR [
 <!ENTITY EpeeDeBois
     SYSTEM "http://epee-de-bois.fr/EDB.xml">
 <!ENTITY CineMarseille
     SYSTEM "http://cine-marseille.fr/CM.xml">
]>
<MOTEUR>
  <CINEMA>
    &EpeeDeBois;
   </CINEMA>
  <CINEMA>
    &CineMarseille;
   </CINEMA>
</MOTEUR>
```

Gestion de l'information avec XML



Les applications du Web avec XML

Publication d'informations: transformation de XML en différents formats/dialectes : HTML, SMIL, WML, SVG, MathML

Diffusion d'informations: Web Sémantique : RDF/RSS, Dublin core

Échange et partage d'informations : dialectes XML partagés entre les membres d'une communauté, services Web (SOAP)

Recherche d'information : langages requêtes (XQuery)

Récapitulons !

XML = format d'échange de données entre application

- Permet de définir des « langages » pour décrire des données (« méta-langage »)
- Permet de représenter pratiquement toute information semi-structurée
- De nombreux outils d'analyse, *parsing*, interrogation, transformation, ...

=> Bien adapté au web.

Les API DOM et Sax

Programmer avec XML

Les deux principales interfaces de programmation XML :

- DOM (*Document Object Model*), basé sur une représentation hiérarchique
- SaX (*Simple API for XML*), basé sur des déclencheurs (événements/action)

Les API XML

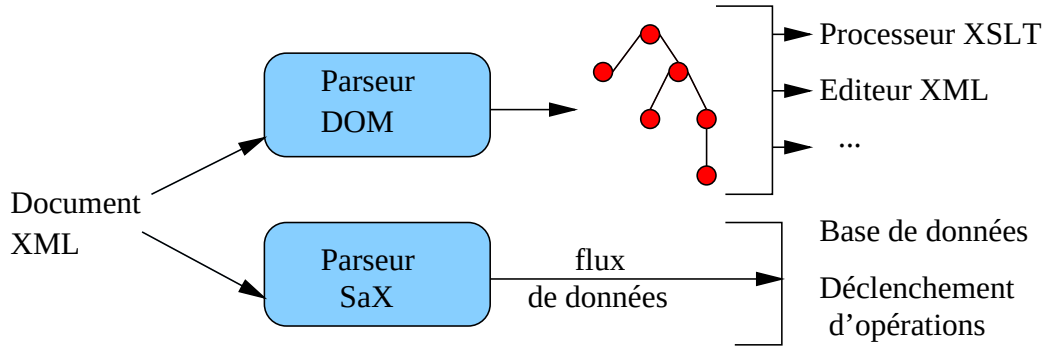
L'API DOM :

- Construit une représentation du document en mémoire sous forme d'arbre
- Adaptée aux applications qui modifient ou traitent dans leur globalité un document.

L'API SaX :

- Définit des *triggers* qui se déclenchent sur certaines balises.
- Adaptée aux applications qui extraient de l'information d'un document

Toutes les applications XML passent par une phase préalable d'analyse



Document Object Model (DOM)

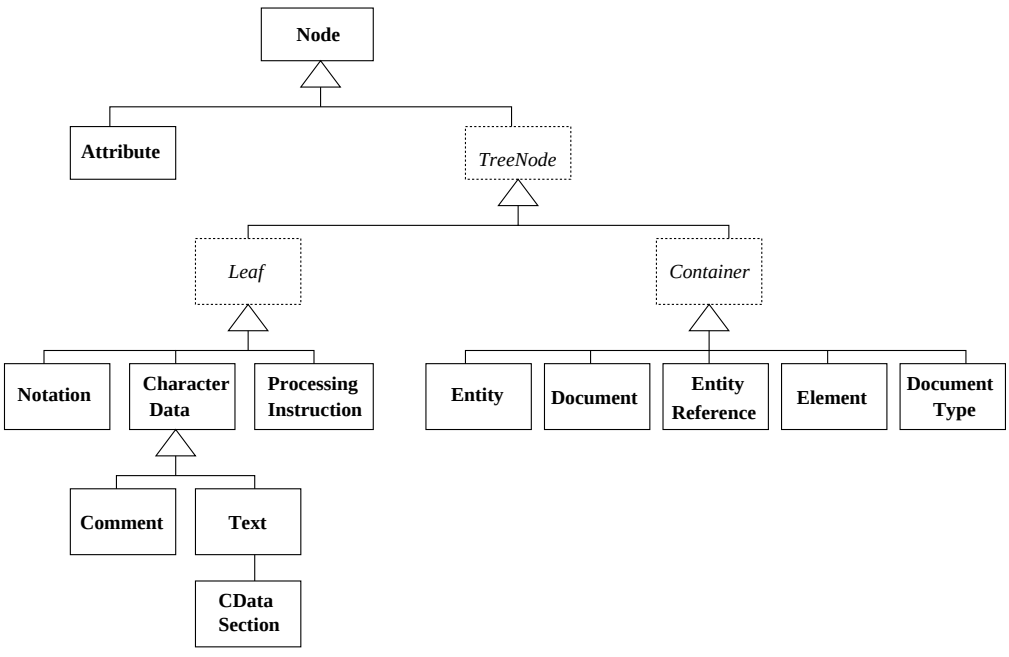
L'API orientée-objet DOM

Un parseur DOM prend en entrée un document XML et construit un **arbre** formé **d'objets** :

- chaque objet appartient à une sous-classe de `Node`
- des opérations sur ces objets permettent de **créer** de nouveaux nœuds, ou de **naviguer** dans le document

=> éditeurs XML, processeurs XSLT

Les nœuds DOM



Types de noeuds DOM

Type de Nœuds	Catégorie syntaxique XML
Document	document XML (racine)
DocumentType	type du document (DTD)
ProcessingInstruction	instruction de traitement
Element	élément XML
Attribute	attribut XML
Entity	déclaration d'entité

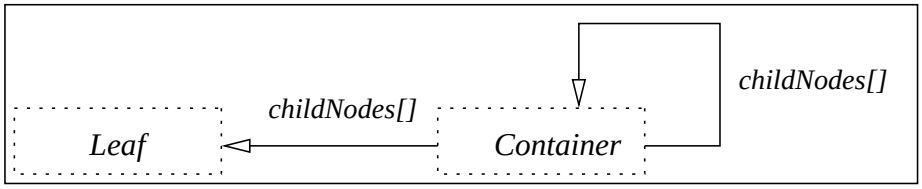
Table 1: Types de nœuds DOM

Types de noeuds DOM (suite)

Type de Nœuds	Catégorie syntaxique XML
EntityReference	référence vers entité
Comment	commentaire
CharacterData	commentaire et section de texte
Text	section de texte
CDataSection	section CDATA
DocumentFragment	fragment de document XML
Notation	notation

Table 2: Types de nœuds DOM

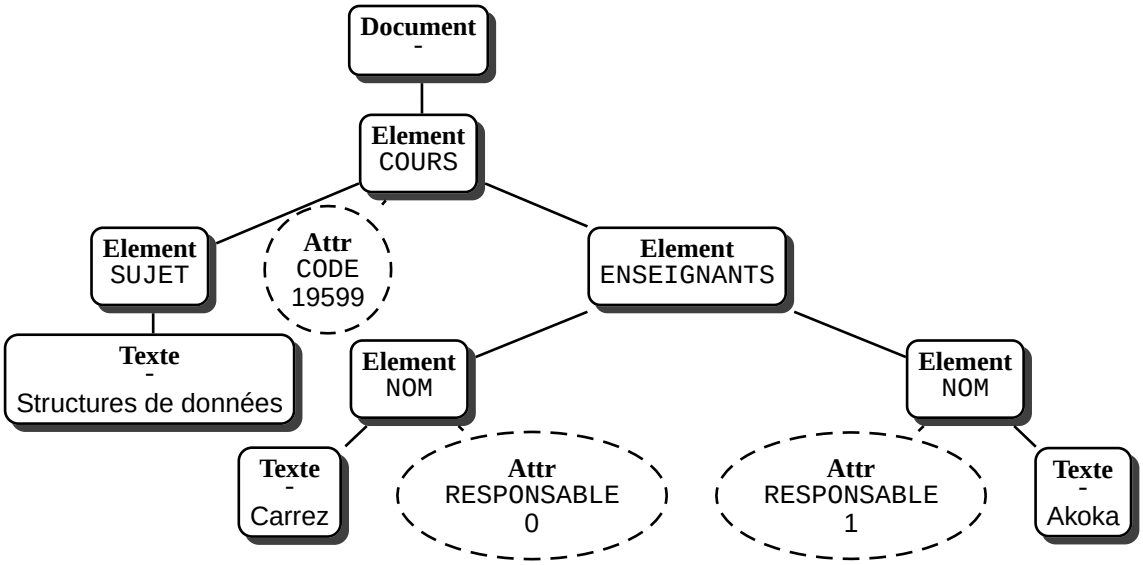
Structure d'un arbre DOM



Un document avec attributs

```
<?xml version='1.0' encoding="ISO-8859-1"?>
<COURS CODE="19599">
  <SUJET>Structures de données</SUJET>
  <ENSEIGNANTS>
    <NOM RESPONSABLE="0">Carrez</NOM>
    <NOM RESPONSABLE="1">Akoka</NOM>
  </ENSEIGNANTS>
</COURS>
```

L'arbre DOM



Propriétés du type Node

Propriété	Type
nodeType	**unsigned short**
nodeName	**DOMString**
nodeValue	**DOMString**
attributes	**NamedNodeMap**

Table 3: Propriétés du type **Node**

Propriétés du type Node

Propriété	Type
parentNode	**Node**
firstChild	**Node**
lastChild	**Node**
childNodes	**NodeList**
previousSibling	**Node**
nextSibling	**Node**

Table 4: Propriétés du type **Node**

Opérations du type Node

Résultat	Méthode	Paramètres
Node	`insertBefore()`	**Node** *nouv*, **Node** *fils*
Node	`replaceChild()`	**Node** *nouv*, **Node** *anc*
Node	`removeChild()`	**Node** *fils*
Node	`appendChild()`	**Node** *nouv*
boolean	`hasChildNodes()`	
Node	`cloneNode()`	**boolean** *prof*

Table 5: Opérations du type **Node**

DOM : Exemple de code

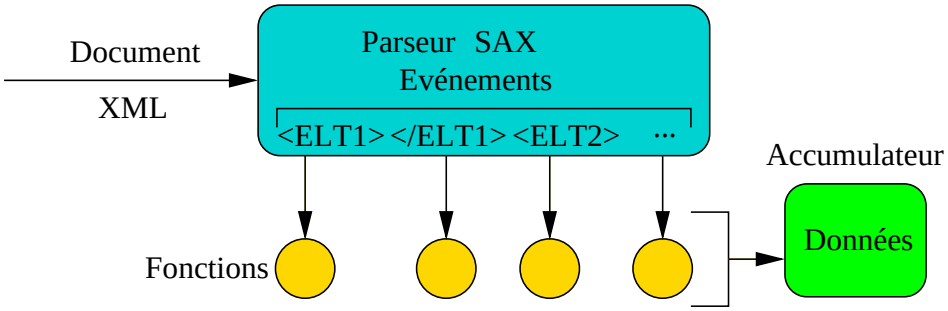
Ajoute
<adventure
type="epic">&adventure1;</adventure>
avant le 4e élément du document :

```
var racine = myDocument.documentElement;
var enfants = racine.childNodes;

var el_nouv = createElement("adventure") ;
var ent_ref = createEntityReference("adventure1");

el_nouv.setAttribute("type", "epic");
el_nouv.appendChild(ent_ref)

insertBefore(el_nouv, enfants.item(3));
```


L’API SAX

Architecture SAX

Associer des événements aux balises :



Les événements SAX

Un parseur SAX génère un événement à chaque fois qu'il rencontre

- le début et la fin de document,
- le début et la fin d'un élément,
- une instruction de traitements
- un commentaire, ...

Les fonctions s'exécutent indépendamment => il faut leur faire partager une zone mémoire (« accumulateur »)

Les événements SAX: exemple

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.apache.xerces.parsers.SAXParser;

public class Trace extends DefaultHandler {
  // début de document
  public void startDocument() {
    System.out.println("start document");
  }
  // fin de document
  public void endDocument() {
    System.out.println("end document");
  }
```


Suite...

```
// balise d'ouverture
public void startElement(String uri,
        String localName, String qName,
        Attributes attributes) {
  System.out.println("starting element: " +
                     qName);
}
// balise de fermeture
public void endElement(String uri,
        String localName, String qName) {
  System.out.println("end element: " + qName);
}
```

Suite...

```
// espace blanc
public void ignorableWhitespace(char[] ch,
        int start, int length) {
  System.out.println("whitespace, length " +
                     length);
}
// instruction de traitement
public void processingInstruction(String target,
        String data) {
  System.out.println("processing instruction: " +
                     target);
}
```


Suite...

```
// texte
public void characters(char[] ch, int start,
        int length){
  System.out.println("character data, length " +
                     length);
}
// procédure principale
public static void main(String[] args) {
  Trace t = new Trace();
  SAXParser p = new SAXParser();
  p.setContentHandler(t);
  try { p.parse(args[0]); }
  catch (Exception e) {e.printStackTrace();}
}
```

Exemple type : insertion dans une BD

À partir d'un flux XML contenant des films :

```
<FILMS>
  <FILM>
    <TITRE>Alien</TITRE>
    <ANNEE>1979</ANNEE>
    <AUTEUR>Ridley Scott</AUTEUR>
    <GENRE>Science-fiction</GENRE
    <PAYS>USA</PAYS>
  </FILM>
</FILMS>
```

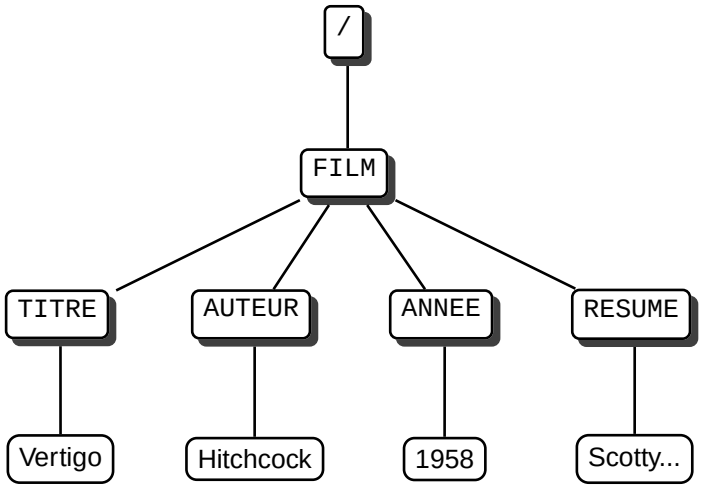

Les événements

- Début de document : connexion à la base
- Balise <FILM> : création d'un enregistrement *Film*
- Balise <TITRE> : on affecte *Film.titre*
- Balise <ANNEE> : on affecte *Film.annee*
- etc..
- Balise <FILM> : insertion de l'enregistrement dans la base
- Balise </FILM> : destruction de *Film*

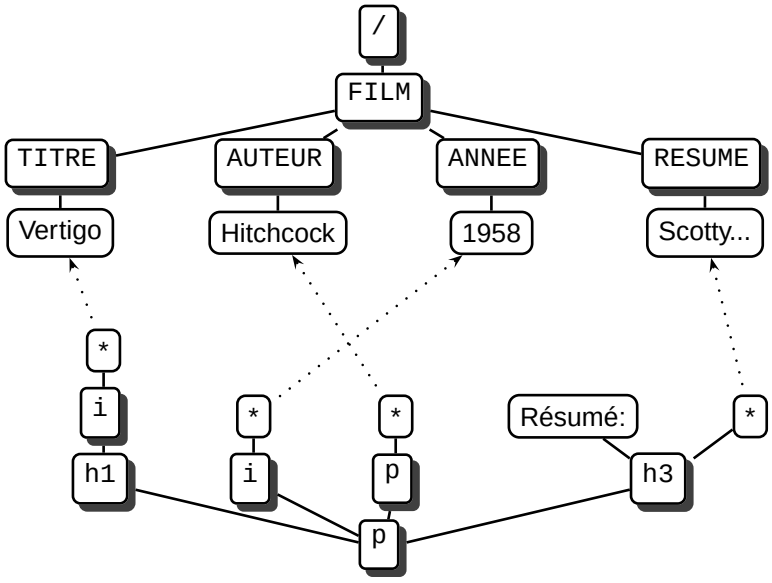
XPath: Sélectionner des Fragments XML

Exemple: Document XML

Document XML:

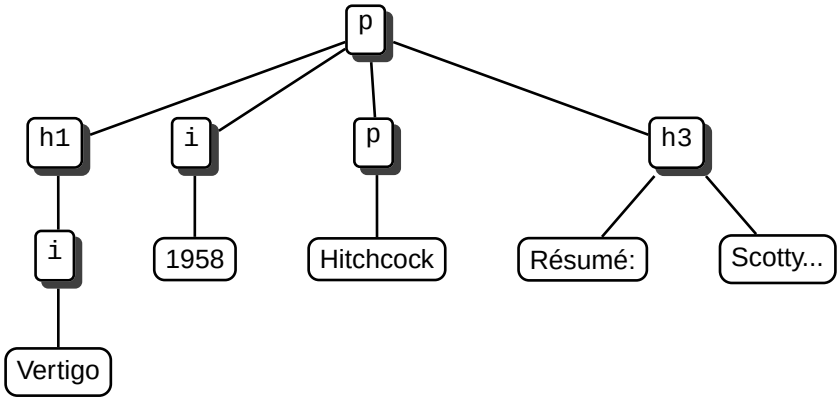


Exemple: Règle de transformation



Exemple: Résultat

Fragment HTML:



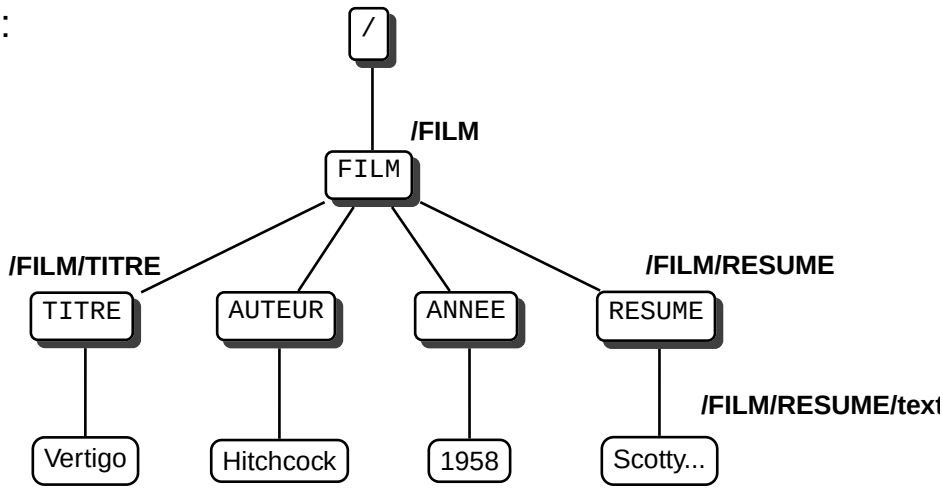
Référencer des fragments XML avec XPATH

Pour pouvoir transformer un document XML il faut pouvoir extraire des fragments (nœuds) XML d'un document $\Rightarrow$ XPath :

- XPath est fondé sur une représentation arborescente (DOM) du document XML
- Objectif : référencer nœuds (éléments, attributs, commentaires, ...) dans un document XML

XPath: Exemples de chemins XPath

Document XML:



XPath: Utilisation

Le langage permet de désigner un ou plusieurs nœuds dans un document XML, à l'aide **d'expressions de chemin**.
XPath est utilisé par

- XSLT pour sélectionner des règles de transformation
- XQuery pour l'interrogation de documents XML
- XML Schéma pour créer des clés et références
- XLink pour créer des liens entre documents/fragments XML

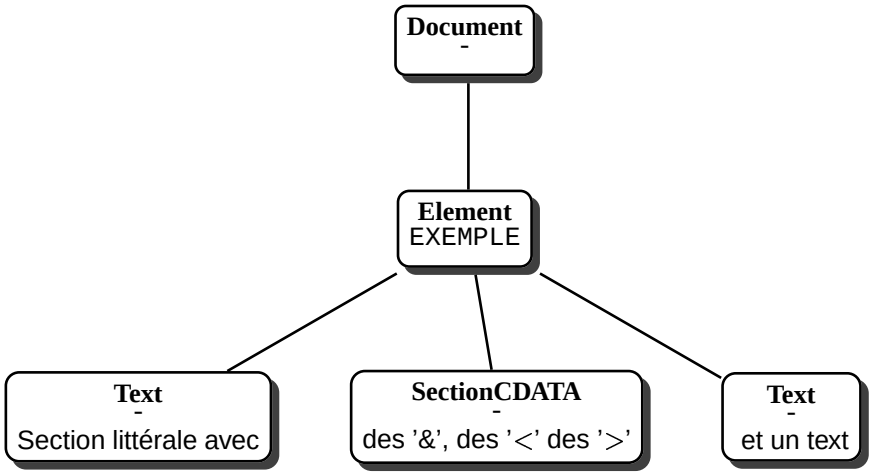
Les arbres XPath

Un typage simplifié par rapport à celui de DOM

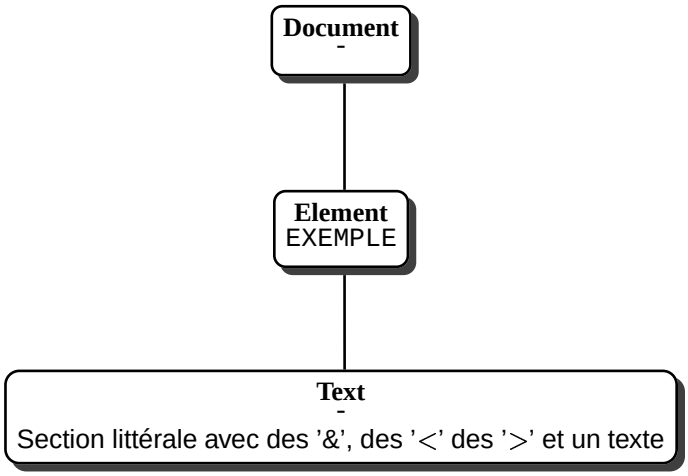
- **Document**
- **Element**
- **Text**
- **Attribut**
- **ProcessingInstruction**
- **Comment**

$\Rightarrow$ pas d'entité, pas de section littérale

Exemple : l'arbre DOM...



Puis l'arbre XPath...



Expressions XPath

Une expression XPath :

- s'évalue en fonction d'un **nœud contexte**
- désigne un ou plusieurs chemins dans l'arbre à partir du nœud contexte
- a pour résultat
  - ▷ un ensemble de nœuds
  - ▷ ou une valeur, numérique, booléenne ou alphanumérique

Chemins XPath

Un chemin XPath est une suite **d'étapes** :

$$[/]\texttt{étape}_1/\texttt{étape}_2/.../\texttt{étape}_n$$

Deux variantes : Un chemin peut être

- **absolu** : `/FILM/RESUME`
  Le nœud contexte est alors la racine du document.
- ou **relatif** : `RESUME/text()`
  Le nœud contexte est un nœud dans le document (pas forcément la racine).

Étapes XPath

Une étape : trois composants

[axe::]filtre[prédicat1][prédicat2]...

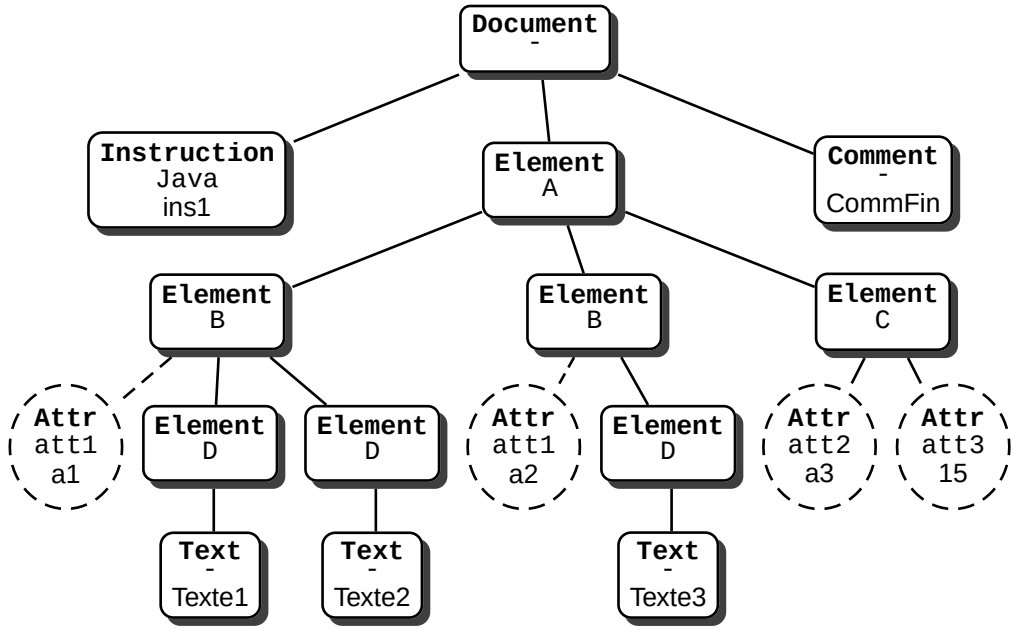
- *l'axe* : sens de parcours des nœuds (par défaut: *child*).
- *le filtre* : type des nœuds/noms des éléments qui seront retenus
- *le(s) prédicat(s)* que doivent satisfaire les nœuds retenus

Les axes XPath

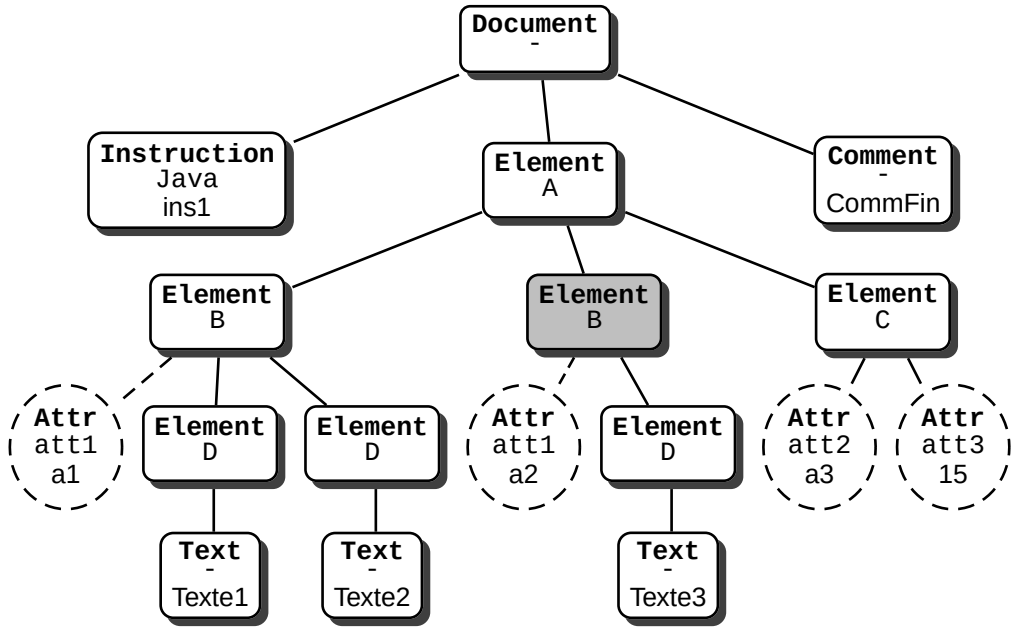
Un axe XPath recouvre les deux notions suivantes :

- un sous-ensemble des nœuds de l'arbre relatif au nœud contexte ;
- l'ordre de parcours de ces nœuds à partir du nœud contexte.

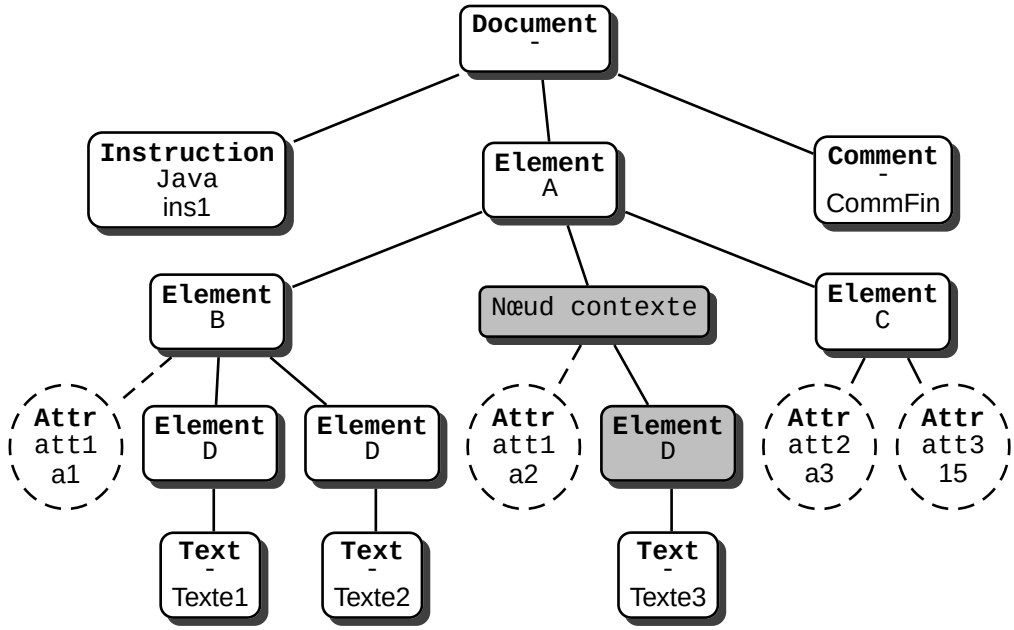
Exemple de référence



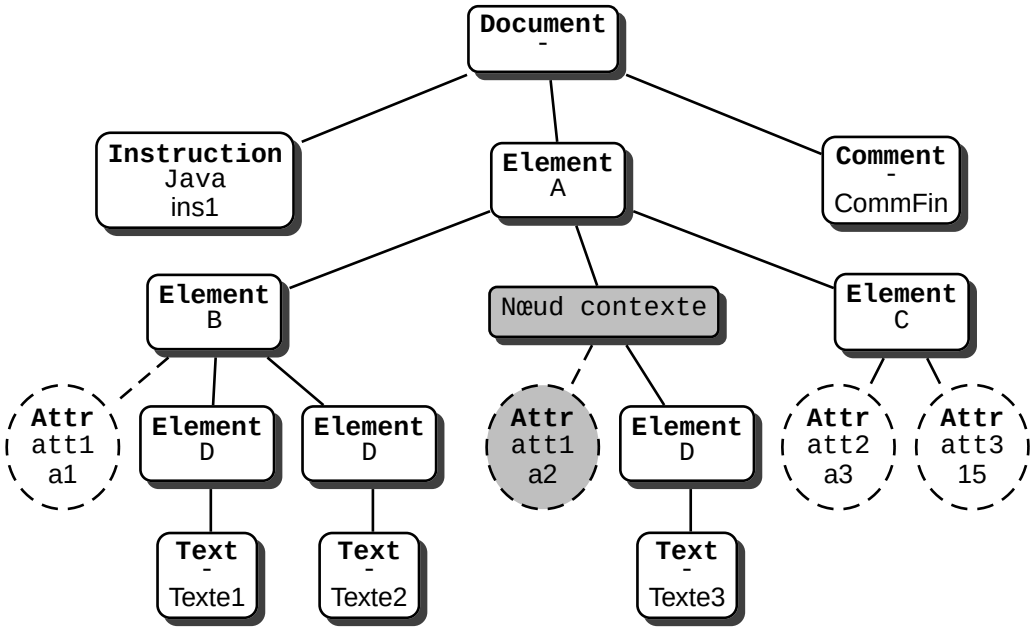
Exemple : le nœud contexte



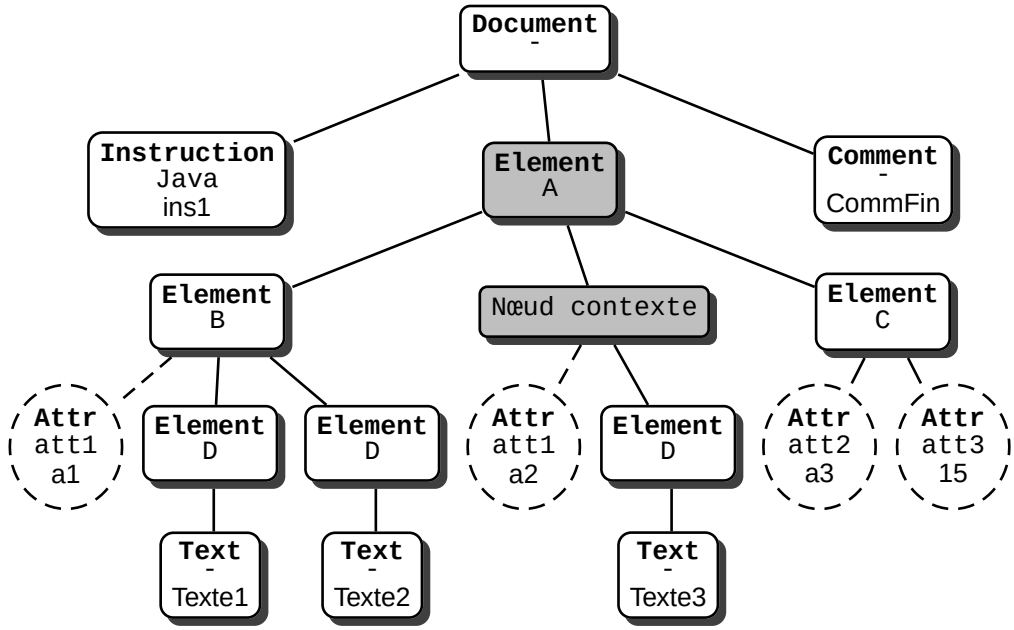
child::D ou D



attribute::att1 ou @att1



parent::A



Notation abrégée de parent::A

La notation abrégée `..` désigne le père du nœud contexte, **quel que soit son type**.

Équivalent à :

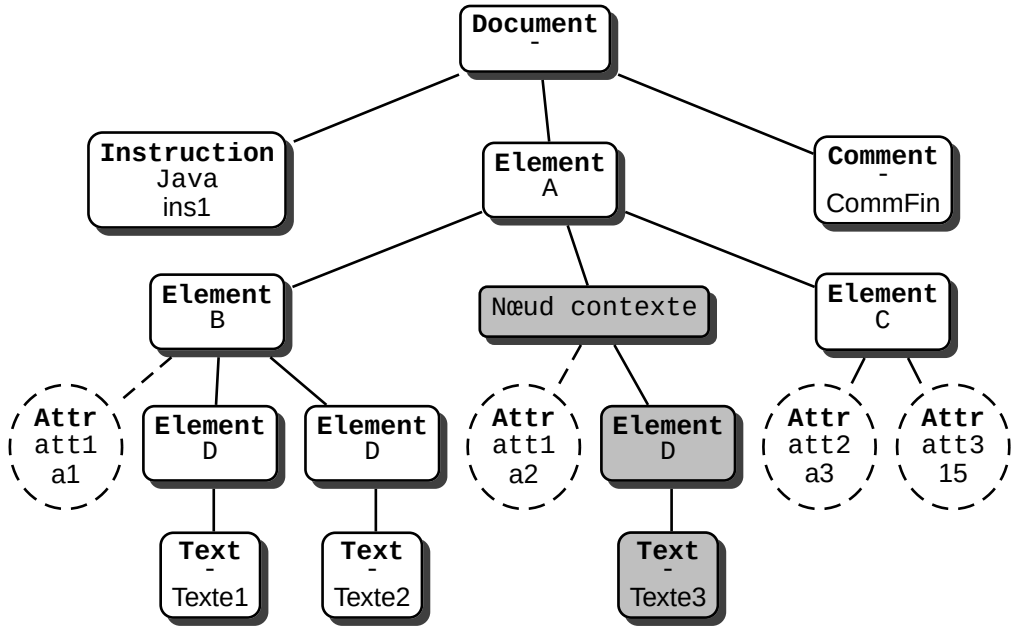
```
parent::node()
```

`node()` est un **filtre** qui désigne tous les types de nœuds (sauf les attributs).

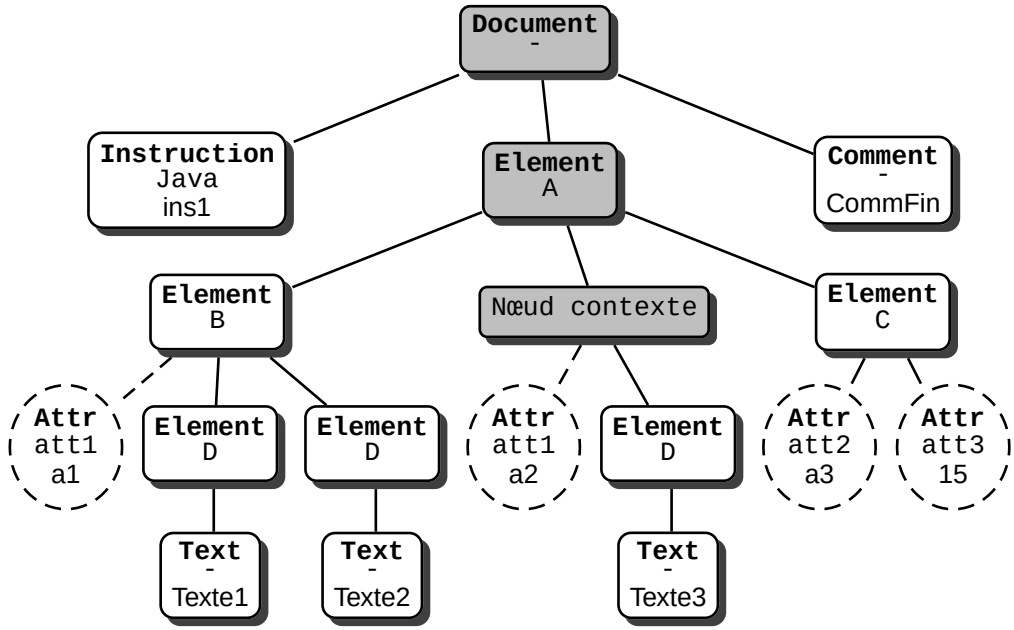
```
parent::*
```

`*` est un **filtre** qui désigne tous les *éléments*.

descendant::node()



ancestor::node()



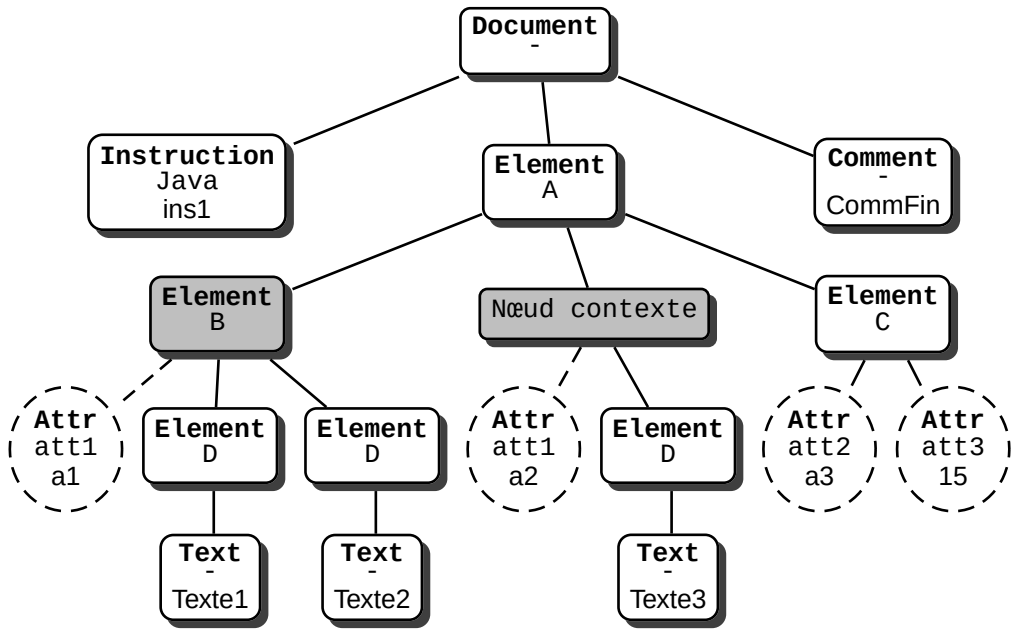
self::node()

- il désigne le nœud contexte lui-même. Doit être complété par un filtre. Exemples :
  - ▷ notre nœud contexte : `self::B`
  - ▷ mais `self::A` renvoie un ensemble vide
- Pour prendre le nœud **quel que soit son type** :

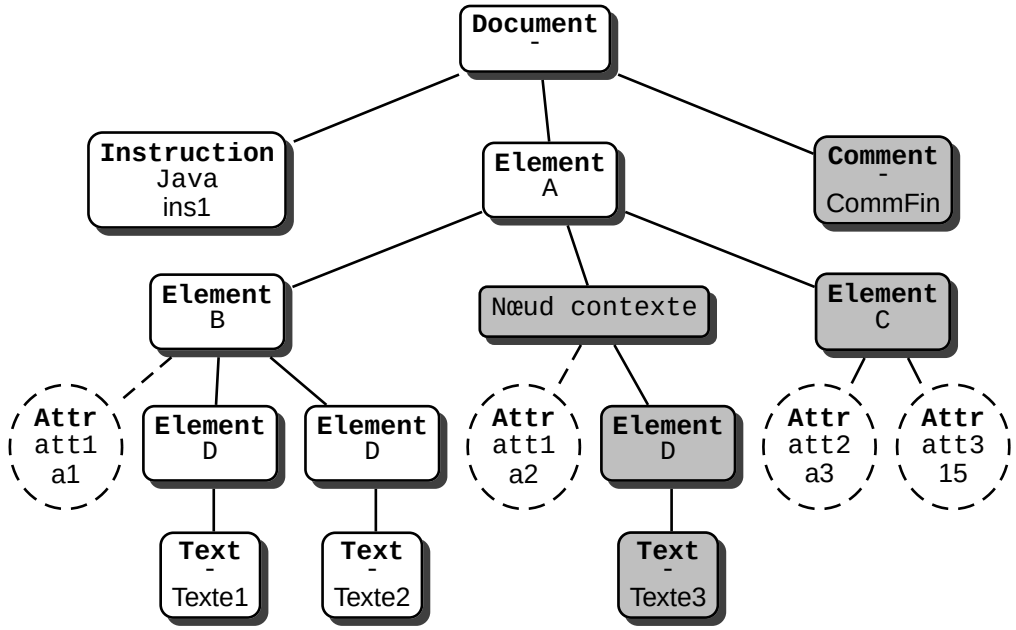
```
self::node()
```

- Notation abrégée : « . »

preceding-sibling::node()



following::node()



Axes: Résumé

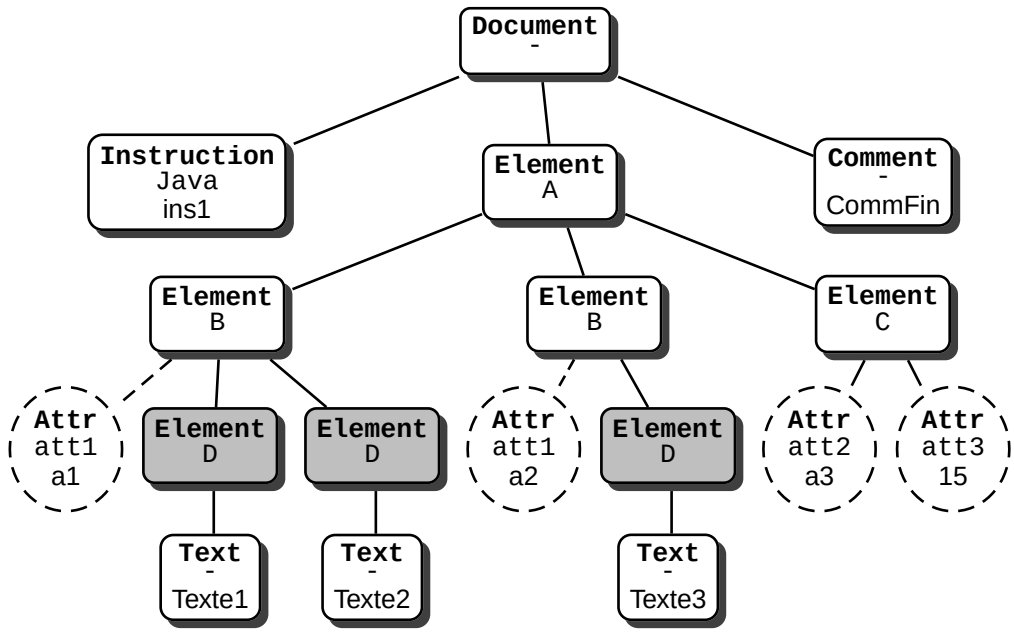
- nœuds enfants et descendants: `child`, `descendant`, `descendant-or-self`
- nœud parent et ancêtres: `parent`, `ancestor`
- frères: `preceding-sibling`, `following-sibling`
- nœuds précédents et suivants: `preceding`, `following`
- nœud même: `self`

Filtres

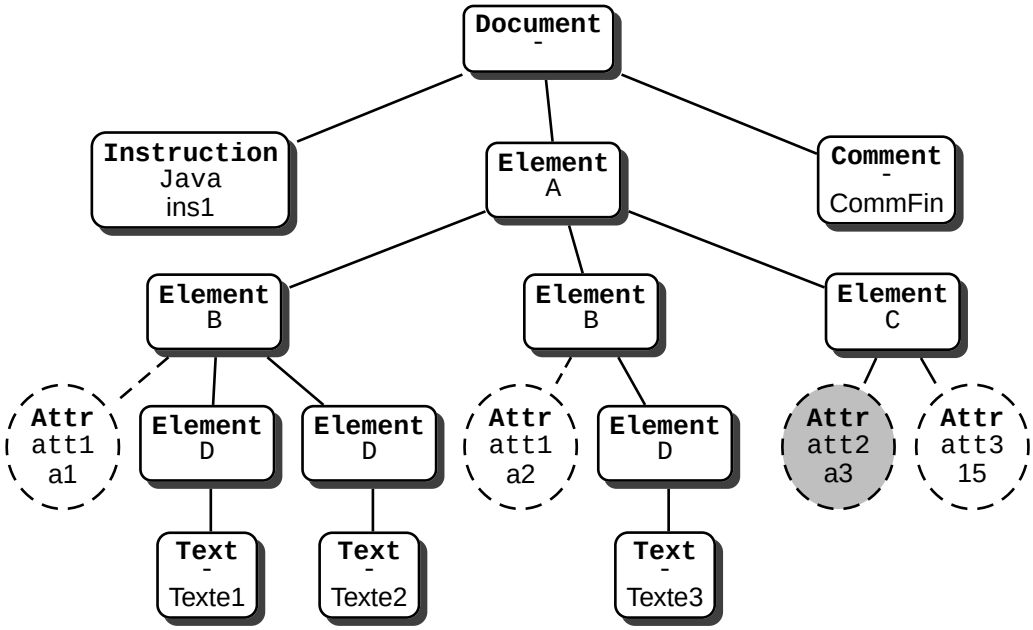
Deux manières de filtrer les nœuds :

- par leur nom :
  - ▷ possible pour les types de nœuds qui ont un nom : **Element**, **ProcessingInstruction** et **Attr**
- par leur type DOM.

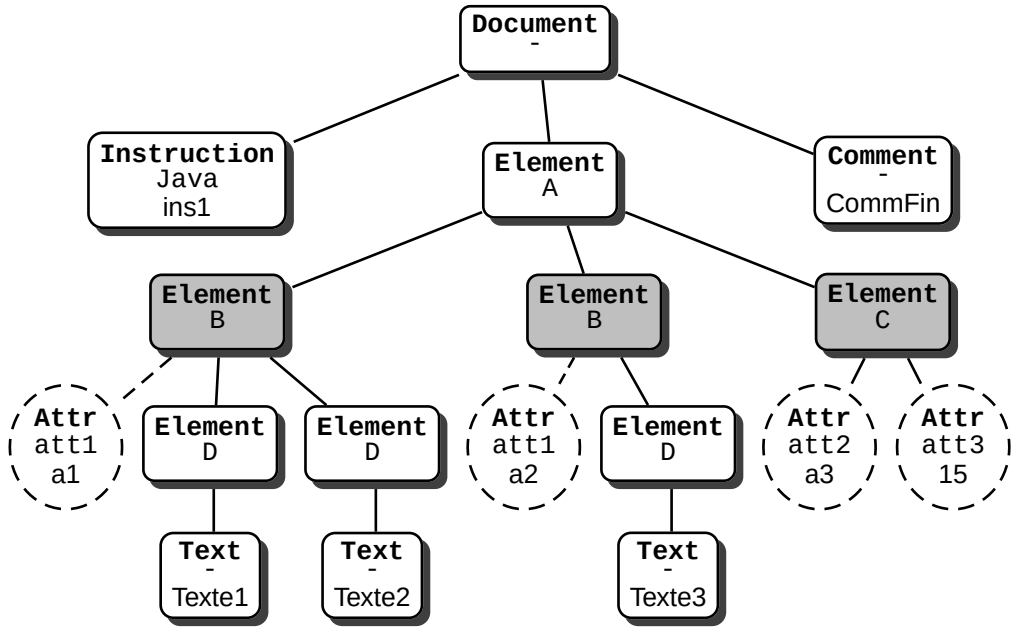
Chemin absolu : /A/B/D



/descendant::node()/@att2



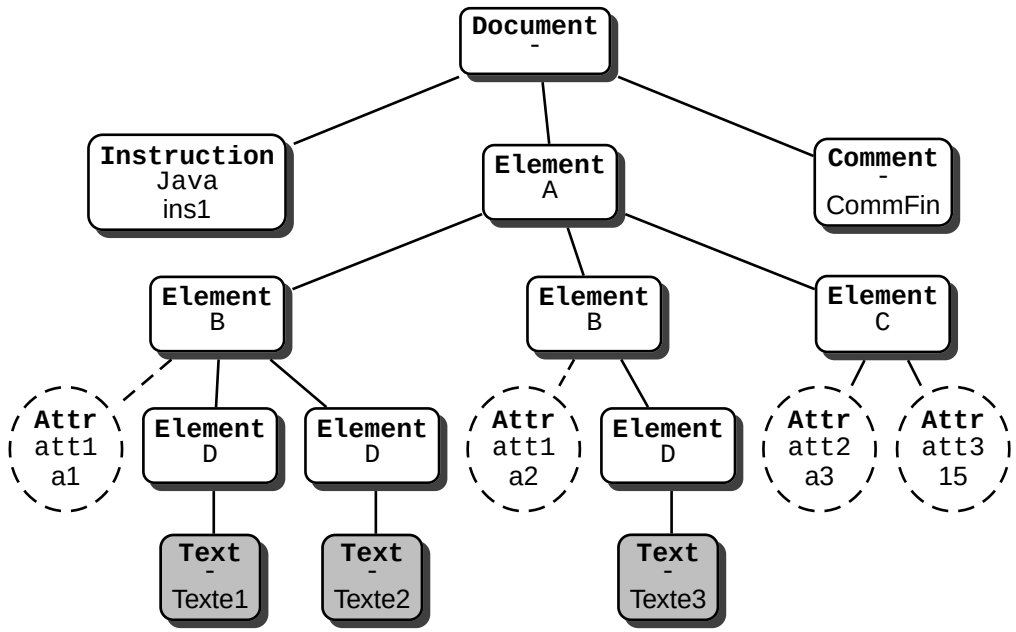
Nom générique : /A/*



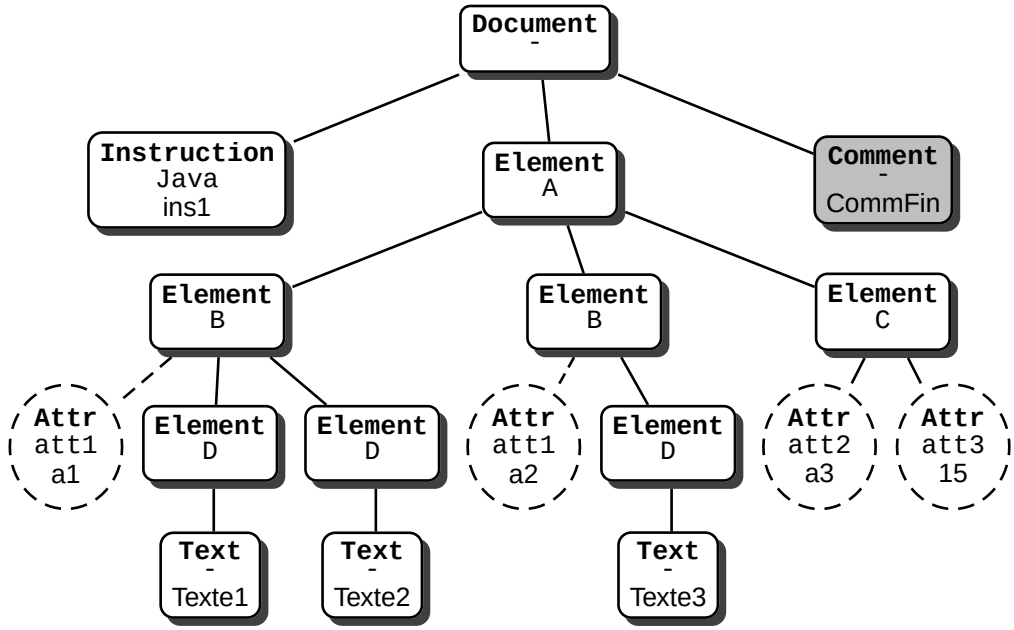
Filtrage sur le type de nœud

- `text()` : Nœuds de type **Text**
- `comment()` : Nœuds de type **Comment**
  Exemple : `/comment()`
- `processing-instruction()` : Nœuds de type **ProcessingInstruction**
  Exemple : `/processing-instruction()`, ou `/processing-instruction('java')`
- `node()` : Tous les types de nœud

/A/B//text()



/comment()



Prédicats

- **Prédicat** : expression booléenne constituée d'un ou plusieurs tests, composés avec les connecteurs logiques habituels `and` et `or`
- **Test** :
  - ▷ toute expression XPath, dont le résultat est convertie en booléen;
  - ▷ une comparaison, un appel de fonction.

⇒ il faut connaître les règles de conversion

Quelques exemples

`/A/B[@att1]:` Les nœuds `/A/B` qui ont un attribut `@att1`

`/A/B[@att1='a1']:` Les nœuds `/A/B` qui ont un attribut `@att1` valant 'a1'

`/A/B/descendant::text()[position()=1]:` Le premier nœud de type **Text** descendant d'un `/A/B`.

`/A/B/descendant::text()[1]:` idem

Pour bien comprendre

Dans l'expression `/A/B[@att1]` :

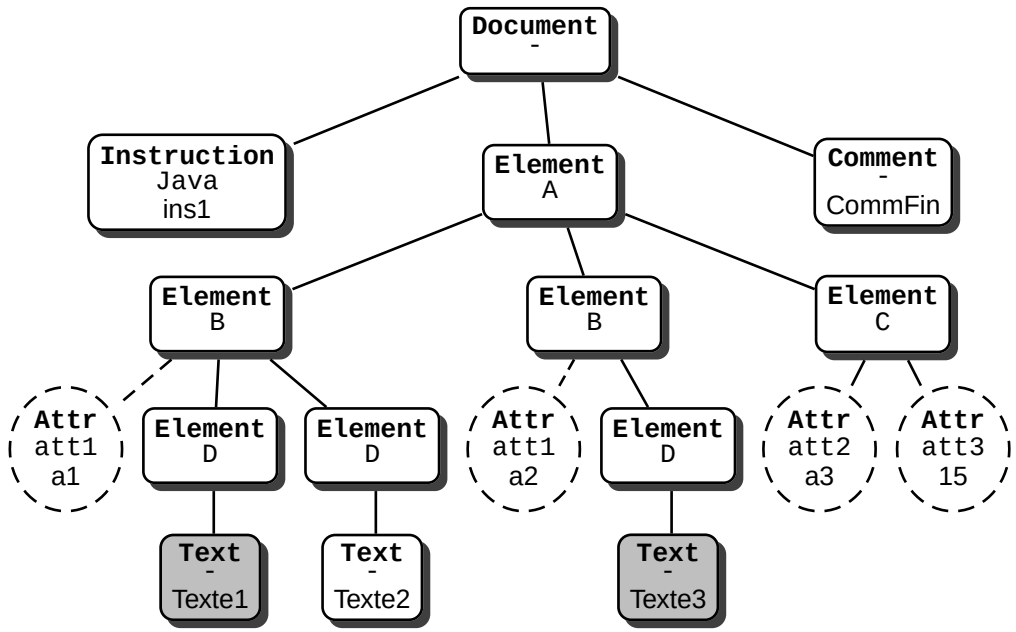
- On s'intéresse aux nœuds de type `B` fils de l'élément racine `A`.
- Parmi ces nœuds on ne prend que ceux pour lesquels le prédicat `[@att1]` s'évalue à `true`
- Cette expression s'évalue avec pour nœud contexte un élément `B`
- `[@att1]` vaut `true` ssi `@att1` renvoie un ensemble de nœuds **non vide**

Contexte d'évaluation

Une étape s'évalue en tenant compte d'un **contexte** constitué de

- un nœud contexte, position initiale du chemin ;
- ce nœud fait lui-même partie d'un ensemble obtenu par évaluation de l'étape précédente
  - ▷ on connaît la **taille** de cet ensemble (fonction *last*())
  - ▷ on connaît la **position** du nœud contexte dans cet ensemble (fonction *position*())

/A/B/descendant::text()[1]



Typage avec XPath

On peut effectuer des comparaisons, des opérations. Cela implique un **typage** et des conversions de type.
Types XPath :

- *les numériques*
- *les chaînes de caractères*
- *les booléens* (`true` et `false`)
- enfin *les ensembles de nœuds*

Numériques

Notation décimale habituelle

- Comparaisons habituelles (<, >, !=)
- Opérations : +, -, *, div, mod
- La fonction *number*() permet de tenter une conversion
- Si la conversion échoue on obtient `NaN` (*Not a Number*). **À éviter...**

Ex : `//node()[number(@att1) mod 2=1]`

Conversions

Deux conversions sont toujours possibles.

- **Vers une chaîne de caractères**.
  - ▷ utile pour la production de texte en XSLT (balise `xsl:value-of`)
- **Vers un booléen**
  - ▷ utile pour les tests effectués dans XSLT (`xsl:if`, `xsl:when`)

Conversions booléennes

- **Pour les numériques** : 0 ou `NaN` sont `false`, tout le reste est `true`
- **Pour les chaînes** : une chaîne vide est `false`, tout le reste est `true`
- **Pour les ensembles de nœuds** : un ensemble vide est `false`, tout le reste est `true`

Comparaisons : des règles de conversion étranges...

Fonctions XPath

Quelques fonctions utiles dans les prédicats :

- *concat(chaîne1, chaîne2, ...)* pour concaténer des chaînes
- *contains(chaîne1, chaîne2)* teste si *chaîne1* contient *chaîne2*
- *count (expression)* renvoie le nombre de nœuds désignés par *expression*
- *name()* renvoie le nom du nœud contexte
- *not(expression)* : négation

Prédicats: Exemples

Axes « d'avancement » :

- `child::*[3]`: le 3e enfant
- `child::*[position()=3]`: idem
- `child::*[last()]`: le dernier enfant
- `descendant::*[last()]`: le dernier descendant

La position d'un nœud dépend de l'axe choisi.

Prédicats: Axes inverses

Axes « inverses » :

- `ancestor::*[1]`: le premier ancêtre du nœud contexte (dernier dans l'ordre du document).
- `preceding-sibling::*[last()]`: le dernier frère qui précède le nœud contexte (premier dans l'ordre du document).

Prédicats: Exemples

Test du nombre d'occurrences :

- `CINEMA[count(SEANCE) > 1]`: cinémas avec au moins 2 séances
- `FILM[count(ACTEUR) = 0]`: films sans acteur
- `FILM[not(ACTEUR)]`: films sans acteur

Prédicats: Sélection par valeur

Sélection par valeur:

- `FILM[not(ACTEUR[NOM='Willis'])]`: films sans Bruce Willis
- `FILM[ACTEUR/NOM='Willis']`: films avec Bruce Willis
- `FILM[ACTEUR[NOM='Willis']]`: idem

Prédicats: Exemples

Test sur la structure : chemins imbriqués avec connecteurs logiques (qualifiers)

- `ACTEUR[NOM and DATENAISSANCE]` ou `ACTEUR[NOM][DATENAISSANCE]`: les acteurs avec un nom et une date de naissance
- `FILM[@TITRE = 'Brazil' and ACTEUR/NOM = 'De Niro']`: le film Brazil avec l'acteur De Niro
- `FILM[@TITRE = 'Brazil'][ACTEUR/NOM = 'De Niro']`: idem

Remarque sur les prédicats

Attention: Les deux premières expressions sont equivalentes, mais pas la troisième! Pourquoi?

- `FILM[ACTEUR and position()=1]`
- `FILM[position()=1][ACTEUR]`
- `FILM[ACTEUR][position()=1]`

Un cas épineux : les espaces

Un document XML avec espaces

```
<?xml version="1.0"
      encoding="ISO-8859-1"?>
<?xml-stylesheet href="ex.xsl"
                 type="text/xsl"?>

<!-- commentaire -->

<!DOCTYPE A SYSTEM "ex.dtd" >
<A>
 <B at1=' val1 '
    at2=' avec blanc '
 />
 <C> </C>
 <D> texte </D>
</A>
```

Où sont les espaces ?

Un peu partout !

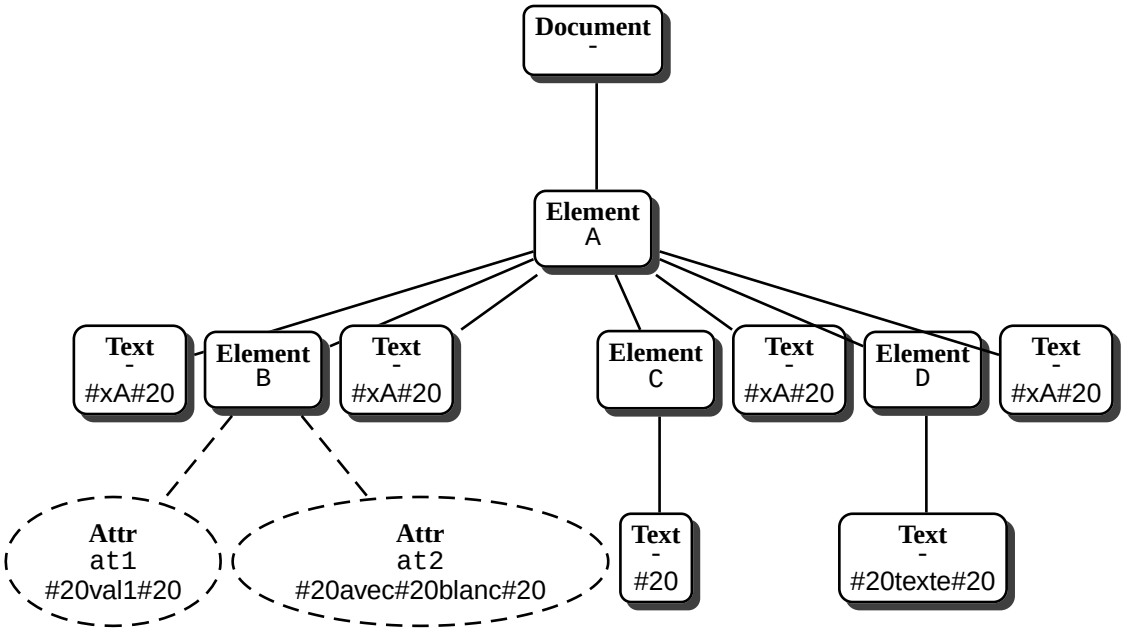
- avant la balise ouvrante de l'élément racine ;
- à l'intérieur d'une balise ;
- à l'intérieur d'un élément ;
- à l'intérieur de la valeur d'un attribut ;
- entre deux balises ;
- entre deux attributs.

Tous n'ont pas la même importance

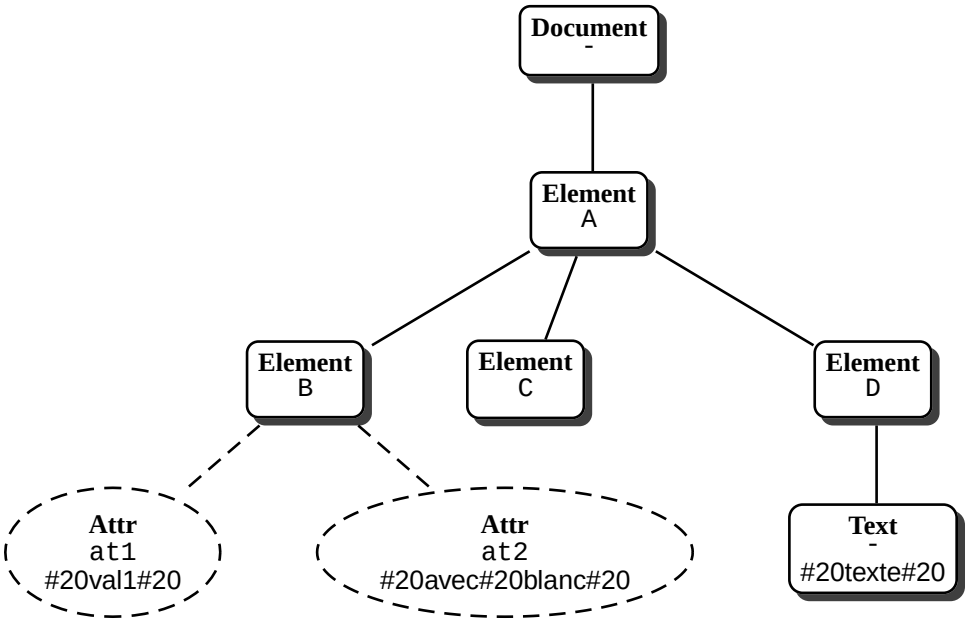
Les espaces dans XPath

- Les espaces du prologue ou de l'épilogue sont supprimés.
- Les espaces consécutifs sont réduits à un seul.
- Les fins de lignes sont représentées par le caractère #xA.
- En général, les espaces en début et fin d'attributs sont supprimées.

L'arbre XPath du document



Instruction `xsl:strip-space`



XPath : Résumé

XPath est un langage pour extraire des noeuds dans un arbre XML :

- On navigue dans l'arbre grâce à des axes de navigation.
- Un chemin de navigation est une séquence d'étapes.
- Dans chaque étape on choisi un axe, un filtre et éventuellement des prédicats.
- Le résultat d'une étape (d'une séquence d'étapes) est un séquence de noeuds.

Introduction à XSLT

Sommaire

Une introduction à XSLT, destinée à comprendre les **mécanismes** du langage.

- Règles XSLT
- Désignation de fragments XML
- Appels de règles
- Application : XML -> HTML et XML -> WML
- Passage de paramètres

Qu'est-ce qu'on peut faire avec XSLT?

Transformer un document XML en

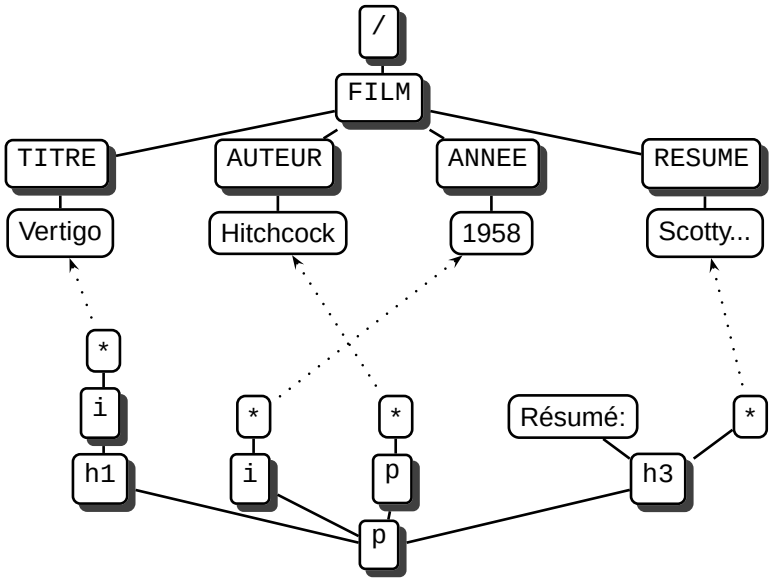
- un ou plusieurs documents XML, HTML, WML, SMIL
- document papier: PDF (XSL-FO), LaTeX
- texte simple

Fonctions d'une Feuille de Style XSLT

Fonction de Base: Langage de règles de transformation de documents/arbres XML:

- extraction de données
- génération de texte
- suppression de contenu (noeuds)
- déplacer contenu (noeuds)
- dupliquer contenu (noeuds)
- trier

Exemple



Exemple: Règle de transformation

```
1  <xsl:template match="FILM">
2    <p>
3      <h1>
4        <i>
5         <xsl:value-of select="TITRE"/>
6        </i>
7      </p>
8      <i>
9        <xsl:value-of select="ANNEE"/>
10     </i>
11     <p>
12       <xsl:value-of select="AUTEUR"/>
13     </p>
14     <h3>Résumé:
15         <xsl:value-of select="RESUME"/>
16     </h3>
17   </p>
18 </xsl:template>
```


Extraction de données

- Exemple : recherche du titre pour le nœud `FILM` :

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

- Plus généralement : on donne un chemin d'accès XPath à un nœud à partir du nœud courant (noeud contexte).

Génération de texte

Produire une phrase quand on rencontre un nœud `FILM` :

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


Génération d'un arbre XML

Produire un document/fragment/arbre XML quand on rencontre un nœud `FILM` :

```
<xsl:template match="FILM">
   <body>
     <p>Un paragraphe</p>
   </body
</xsl:template>
```

Génération avec extraction

Produire un document/fragment/arbre XML quand on rencontre un nœud `FILM` :

```
1 <xsl:template match="FILM">
2    <body>
3      <p>
4        <xsl:text>Titre: </xsl:text>
5        <xsl:value-of select="TITRE"/>
6      </p>
7    </body>
8 </xsl:template>
```


Structure de base : les règles

Règle = *template* : élément de base pour produire le résultat

- une règle s'applique dans le contexte d'un nœud de l'arbre
- l'application de la règle produit un fragment du résultat.

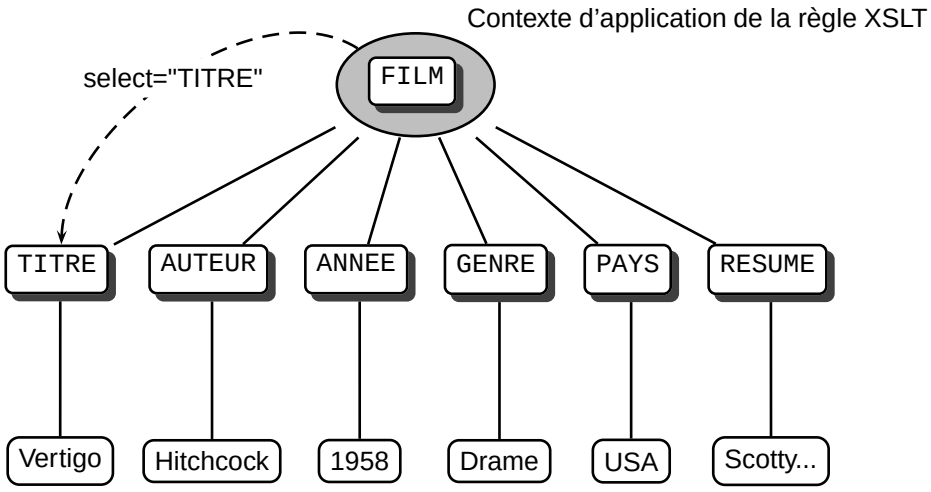
Programme XSLT = ensemble de règles pour construire un résultat

Un exemple

```
1  <xsl:template match="FILM">
2   <p>
3     <h1>
4       <i><xsl:value-of select="TITRE"/></i>
5     </h1>
6     <i><xsl:value-of select="ANNEE"/></i>
7     <p><xsl:value-of select="AUTEUR"/></p>
8     <h3>
9       Résumé: <xsl:value-of select="RESUME"/>
10    </h3>
11   </p>
12 </xsl:template>
```

- **Motif de sélection** : `match="FILM"`
- **Corps de la règle** = fragment d'arbre à produire

Illustration



Une règle complète

```
<xsl:template match="FILM">
 <html>
   <head>
    <title>Film:
      <xsl:value-of select="TITRE"/>
    </title>
   </head>
   <body>
     Le genre du film, c'est
     <b><xsl:value-of select="GENRE"/></b>
   </body>
 </html>
</xsl:template>
```


Le résultat

On obtient :

```
    <html>
      <head>
        <title>Film:
            Vertigo
        </title>
      </head>
      <body>
        Le genre du film, c'est
          <b>Suspense</b>
      </body>
    </html>
```

Chemins complexes

Dans l'exemple précédent, on accédait aux fils d'un nœud.
En fait on peut :

- Accéder à tous les descendants
- Accéder aux parents, aux frères, aux neveux...
- Accéder aux attributs
- Effectuer des boucles

Le langage d'expression de chemins : XPath.

Exemple: Document XML

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2
3  <?xml-stylesheet href="Salle.xsl" type="text/xsl"?>
4  <?cocoon-process type="xslt"?>
5  <SALLE NO='1' PLACES='320'>
6    <FILM>
7      <TITRE>Alien</TITRE>
8      <AUTEUR>Ridley Scott</AUTEUR>
9      <ANNEE>1979</ANNEE>
10     <GENRE>Science-fiction</GENRE>
11     <PAYS>Etats Unis</PAYS>
12     <RESUME>Près d'un vaisseau spatial échoué sur une lointaine
13        planète, des Terriens en mission découvrent de bien étranges
14        "oeufs". Ils en ramènent un à bord, ignorant qu'ils viennent
15        d'introduire parmi eux un huitième passager particulièrement
16        féroce et meurtrier.
17     </RESUME>
18   </FILM>
19   <REMARQUE>Réservation conseillée</REMARQUE>
20   <SEANCES>
21         <SEANCE>15:00</SEANCE>
22         <SEANCE>18:00</SEANCE>
23         <SEANCE>21:00</SEANCE>
24    </SEANCES>
25 </SALLE>
```

Exemple: Traduction de Salle

```
<xsl:template match="SALLE">
 <h2>Salle No
       <xsl:value-of select="@NO"/></h2>
 Film: <xsl:value-of select="FILM/TITRE"/>
 de    <xsl:value-of select="FILM/AUTEUR"/>
 <ol> <xsl:for-each select="SEANCES/SEANCE">
        <li><xsl:value-of select="."/></li>
      </xsl:for-each>
 </ol>
</xsl:template>
```

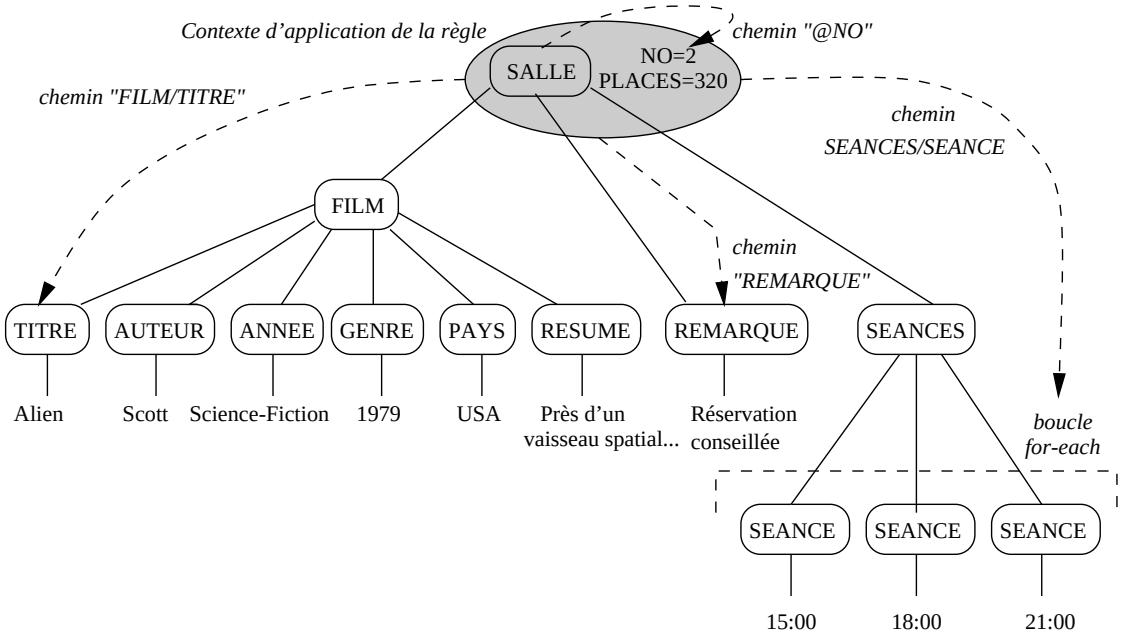

Le résultat

Appliqué à *Salle1.xml* :

```
 <h2>Salle No 1 </h2>
 Film:  Alien
 de     Ridley Scott
 <ol>
        <li>  15:00</li>
        <li>  18:00</li>
        <li>  21:00</li>
 </ol>
```

NB : c'est un **fragment HTML**, à intégrer dans un document complet.

Salles et séances



Chemins complexes

- **Descendants** : le titre du film est désigné par `select="FILM/TITRE"` ;
- **Attributs** : le numéro de la salle est désigné par `select="@NO"` ;
- **Boucles** : avec `xsl:for-each` sur l'ensemble des séances désignées par `select="SEANCES/SEANCE"` ;
- **L'élément contexte** : désigné par le symbole '.', comme dans `select="."`.

Appels de règles

En général on produit un résultat en combinant plusieurs règles :

- La règle initiale s'applique à la racine du document traité ('/')
- On produit alors le **cadre** du document HTML
- On appelle d'autres règles pour compléter la création du résultat

Exemple : L'Épée de bois

« Cadre » HTML, puis appel de la règle `CINEMA`

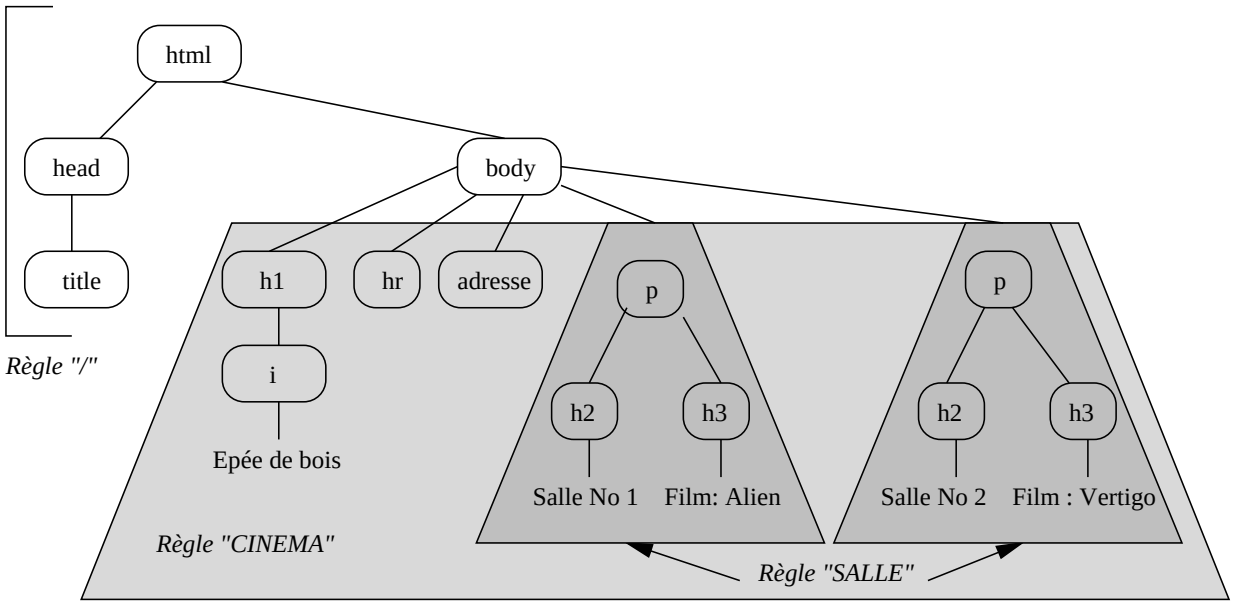
```
<xsl:template match="/">
 <html>
  <head><title>Programme de
          <xsl:value-of select="CINEMA/NOM"/>
        </title>
  </head>
  <body bgcolor="white">
     <xsl:apply-templates select="CINEMA"/>
  </body>
 </html>
</xsl:template>
```

Règle CINEMA

Exploitation de l'élément CINEMA, puis appel à la règle SALLE

```
1  <xsl:template match="CINEMA">
2    <h1><i>
3        <xsl:value-of select="NOM"/>
4    </i></h1><hr/>
5        <xsl:value-of select="ADRESSE"/>,
6    <i>Métro: </i>
7        <xsl:value-of select="METRO"/>
8    <hr/>
9      <xsl:apply-templates select="SALLE"/>
10
11  </xsl:template>
```


Vue d'ensemble

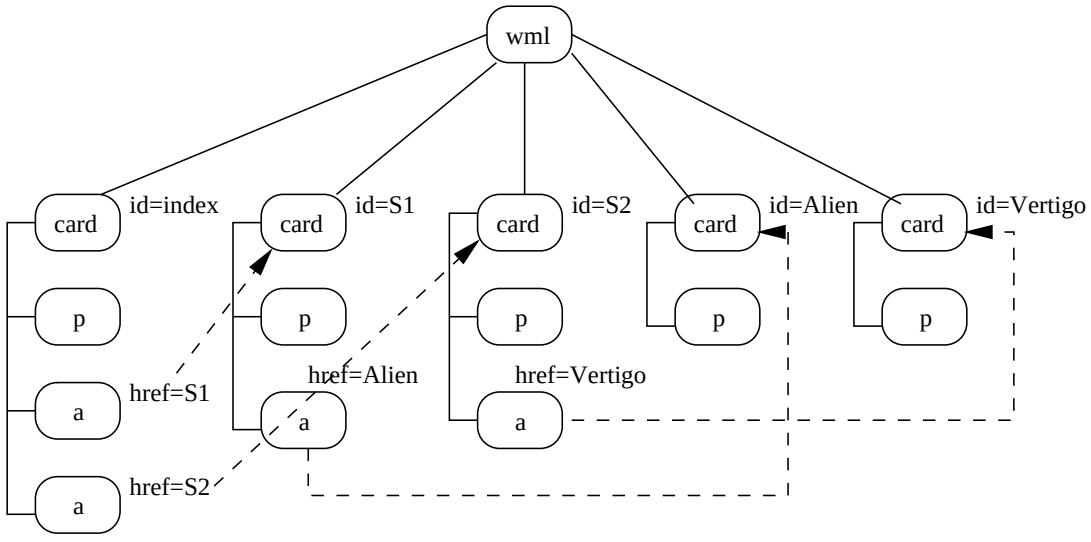


La même chose, en WML

On génère les cartes du site

```
1  <xsl:template match="/">
2   <wml>
3     <!-- Carte d'accueil -->
4     <xsl:apply-templates select="CINEMA"/>
5
6     <!-- Cartes pour les salles et séances -->
7     <xsl:apply-templates select="CINEMA/SALLE"/>
8
9     <!-- création des cartes pour les films -->
10    <xsl:apply-templates select=".//FILM"/>
11  </wml>
12 </xsl:template>
```


Arbre XML du site



Règle CINEMA

```
<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>
```


Résultat

```
  <card id="index" title="Programme">
    <p align="center">
      Ep&#xE9;e de bois
    </p>
    <p>
      <a href="#S1"> Salle 1: </a>
      Alien
    </p>
    <p>
      <a href="#S2"> Salle 2: </a>
      Vertigo
    </p>
  </card>
```

Règle 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>
```


Résultat

```
  <card id="S2">
    <p>
      S&#xE9;ances salle 2
    </p>
    <p>
      <a href="#Vertigo">
          Film : Vertigo</a>
    </p>
    <p>
      S&#xE9;ance 22:00
    </p>
  </card>
```

XSLT avec paramètres

```
<html>
<head>
  <title>Formulaire de Recherche</title>
</head>
<body bgcolor="white">
 <h1>Formulaire de Recherche</h1>
 <form method='get' action='Moteur.xml'
       name='Form'>
 Film: <input type='text' name='titre'> <br>
 Séance: <input type='text' NAME='seance' >
         (hh:mm)<br>
 Ville: <input type='text' name='ville'><br>
 <input type='submit' name='chercher'
        value="Chercher"/>
 </form>
</body>
</html>
```


Le document XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE MOTEUR [
 <!ENTITY EpeeDeBois
     SYSTEM "http://epee-de-bois.fr/EDB.xml">
 <!ENTITY CineMarseille
     SYSTEM "http://cine-marseille.fr/CM.xml">
]>
<MOTEUR>
  <CINEMA>
    &EpeeDeBois;
   </CINEMA>
  <CINEMA>
    &CineMarseille;
   </CINEMA>
</MOTEUR>
```

Traitement des paramètres

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

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


Conclusion sur XSLT

Un langage totalement adapté au traitement de documents XML

- Parcours d'un document, vu comme un arbre
- Déclenchement de règles sur certains nœuds
- Association de plusieurs programmes à un même document

Bibliographie sur XSLT

1. Recommendation XSLT sur le site du W3C
2. B. Amann et P. Rigaux, Comprendre XSLT, O'Reilly
   `http://cortes.cnam.fr:8080/XSLT`
3. P. Wadler, A formal semantics of patterns in XSLT