Programmation XML

Bernd Amann

Objectifs et plan du cours

Objectifs:

- Apprendre les bases de XML
- Etudier et mettre en oeuvre deux langages (XPath et XSLT) conçues pour la programmation XML.
- Site Web: http://cortes.cnam.fr:8080/XSLT
- Livre: B. Amann et P. Rigaux, Comprendre XSLT, O'Reilly

XML par l'éxemple

Exemple : La fiche du film *Gladiator*

La fiche d'un film peut être

- stockée dans une base de données
- publiée en HTML pour Netscape/IE
- publiée en WML pour le portables WAP
- publiée en SMIL pour Realplayer
- référencée dans un moteur de recherche SallesEnLigne.com

L'information ?

- c'est la *même*, sous des formes différentes
- elle est échangée entre plusieurs acteurs (base de données, serveur Web, clients Web, portables...)

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>
```

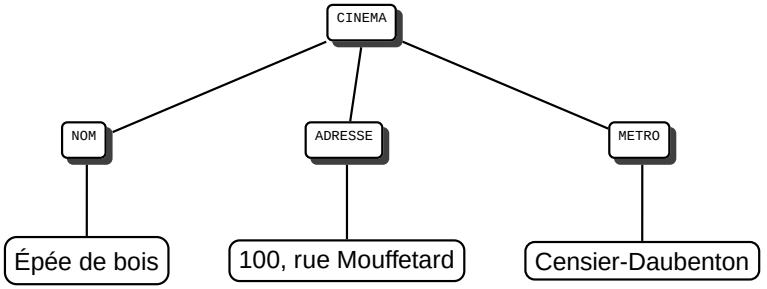

Le même, mieux présenté

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



Documents XML

Qu'est-ce qu'un **document XML** ?

- C'est un **contenu** alphanumérique
- Il est **structuré** avec des balises

Indépendant de la **représentation physique** (origine)

- Un ou plusieurs fichiers ?
- Un message ?
- Un extrait d'une base de données ?
- Tout ça à la fois ...

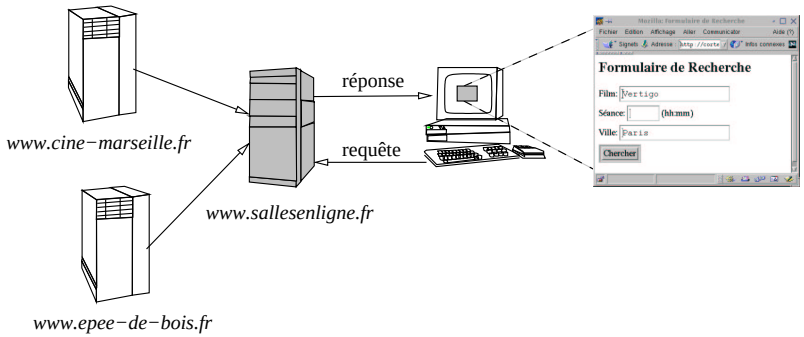
Application XML

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

Exemple : moteur de recherche



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: DTD
- Faire une **traduction** de l'un à l'autre: XSLT

Les DTD

Document Type Definition

- Pour définir la **structure** d'une classe de documents (d'un langage)
- Exemple : un élément de type texte :

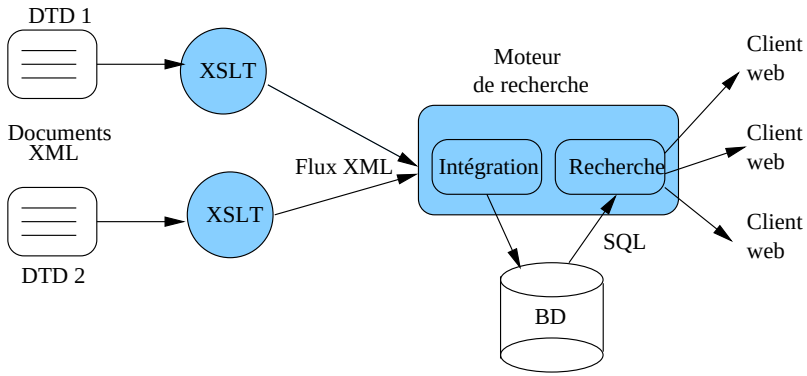
```
<!ELEMENT TITRE ( #PCDATA ) >
```

- Exemple : un élément constitué d'une liste

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

Architecture

(Démo)



Récapitulons !

XML = format d'échange de données entre application

- Permet de définir des « langages » pour décrire des données (« méta-langage »)
- De nombreux outils d'analyse, *parsing*, interrogation, ...
- Transformation d'un langage à un autre avec XSLT

=> Bien adapté au web.

XML (eXtensible Markup Language)

W3C et XML

- Le World Wide Web Consortium (W3C)
  - URL: `http://www.w3.org`
  - 400 partenaires industriels, parmi lesquels les plus grand comme Oracle, IBM, Compaq, Xerox, Microsoft
  - Laboratoires de recherche: MIT pour les États Unis, INRIA pour l'Europe, université Keio (Japon) pour l'Asie
- XML : recommendation W3C pour
  - les documents Web (généralisation de HTML),
  - mais aussi pour l'échange, la transformation, l'intégration et l'interrogation des données sur le Web.

La syntaxe XML

La syntaxe XML permet la représentation d'une information avec :

- des éléments (et leurs attributs)
- des commentaires
- des instructions de traitement
- des sections de texte

La déclaration XML

Tout document XML peut être précédé par une déclaration :

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

- l'attribut `encoding` indique le jeu de caractères utilisé dans le document
- l'attribut optionnel `standalone` indique si le document est composé de plusieurs entités.

Déclaration de type

On peut indiquer qu'un document est conforme à une *DTD*, et déclarer des *entités*.

```
<!DOCTYPE nom SYSTEM "sourceExt" [decLoc]>
```

- *nom* est le type de l'élément racine
- *sourceExt* est un source extérieure contenant la DTD
- *decLoc* sont des déclarations locales (pour les entités principalement)

Entités et références à des entités

Les *entités* servent à factoriser des parties du document.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE A SYSTEM "minimal.dtd" [
  <!ENTITY  monTexte "texte simple">
  <!ENTITY  maSignature SYSTEM "signature.xml">
]>
<A>
  Du &monTexte;, sans caractères réservés:
  ni &lt; ni &gt; ni &amp; ni &apos; ni &quot;
  &maSignature;
</A>
```

Entités caractères

Déclaration entité	Référence	Car.
`<!ENTITY lt "&#60;">`	`<`	<
`<!ENTITY gt "&#62;">`	`>`	>
`<!ENTITY amp "&#38;">`	`&`	&
`<!ENTITY apos "&#39;">`	`&apos;`	'
`<!ENTITY quot "&#34;">`	`&quot`	"

Table 1:

Commentaires et instructions de traitement

- Les commentaires : à utiliser avec parcimonie :
  `<!-- Ceci est un commentaire -->`
- Instructions de traitement : lié au système qui traite le document :
  `<?xml-stylesheet href="prog.xslt" type="text/xslt">`

Éléments

Dans la forme sérialisée :

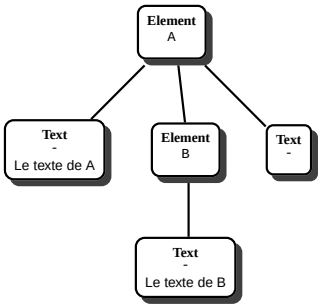
- C'est une **balise ouvrante** avec un nom, puis un **contenu**, puis une **balise fermante**

Dans la forme arborescente

- C'est un nœud avec un nom
- Le contenu est un arbre

Exemple : un élément avec contenu

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<A>Le texte de A
  <B>Le texte de B</B>
</A>
```



Quelques remarques sur les éléments

- Un nom d'élément ne contient pas de blanc, ni de caractère accentué
- Les majuscules sont distinguées des minuscules
- Il existe une forme abrégée pour les éléments sans contenu : `<C></C>` peut s'écrire `<C/>`.
- **Tout document comprend un et un seul élément racine**

Attributs

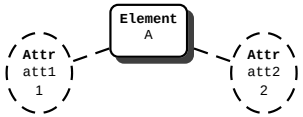
Les attributs constituent un **autre** moyen de représenter de l'information.

`<A att1='1' att2='2'>`

- l'ordre des attributs n'est pas important
- il doit toujours y avoir une valeur, encadrée par des guillemets (différent de HTML)
- il ne peut pas y avoir deux attributs avec le même nom dans un élément.

Attributs : exemples

- `<A att1='1' att2='2'>` est équivalent à `<A att2='2' att1='1'>`



- `<A att=a>` n'est pas bien formé : pas d'apostrophe
- `<A att1='1' att1='2'/>` : interdit
- `<A att1='1'/> <B att1='2'/>` : autorisé (deux éléments différents)

Résumé : structure d'un document XML

Un document XML comprend trois parties :

- le **prologue**, avec la déclaration XML, la DTD, des commentaires, des instructions de traitements (optionnels)
- un **élément racine** avec son contenu
- un **épilogue** avec des commentaires, ou des instructions de traitements (optionnels)

Le contenu du document proprement dit est le contenu de l'élément racine.

Document Object Model (DOM)

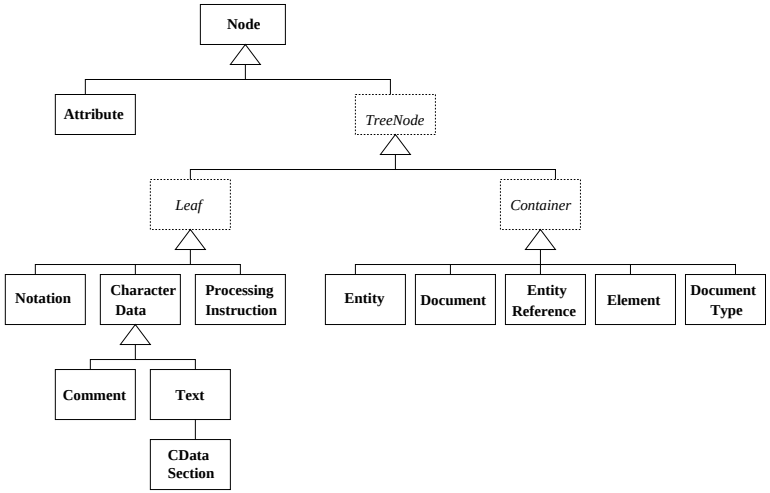
Le modèle DOM

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 types de noeuds DOM



Propriétés d'un Noeud

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

Table 2: Propriétés du type **Node**

Propriétés du type Node

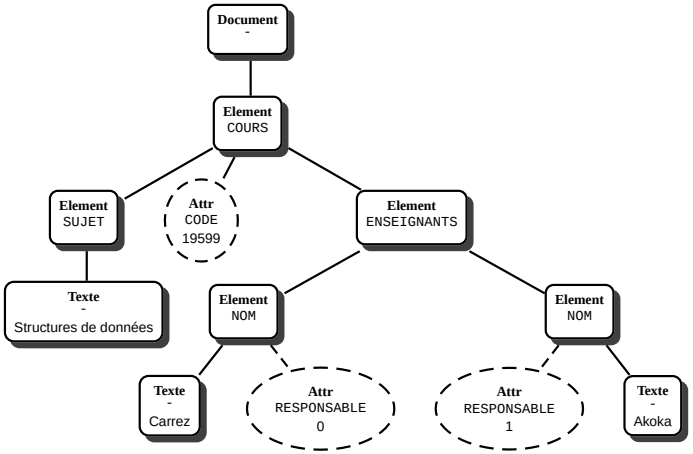
Propriété	Type
parentNode	**Node**
firstChild	**Node**
lastChild	**Node**
childNodes	**NodeList**
previousSibling	**Node**
nextSibling	**Node**

Table 3: Propriétés du type **Node**

Un exemple

```
<?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



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 4: 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));
```


Déclaration du Type de Document (DTD)

Documents XML valides et bien-formés

Document XML **bien-formé**:

- pas de DTD
- la structure est imbriquée (arborescence)

Document XML **valide**:

- respecte une DTD
- respecte *l'intégrité référentielle* :
  - toutes les valeurs d'attributs de type ID sont distinctes
  - toutes les références sont valides

DTD: Exemple

```
1  <!ELEMENT Officiel (#PCDATA | cinéma | film)*>
2  <!ELEMENT cinéma (nom, adresse, (séance)*)>
3  <!ELEMENT nom (#PCDATA) >
4  <!ELEMENT adresse (ville, rue, (numéro)?)>
5  <!ELEMENT séance EMPTY>
6  <!ATTLIST séance heure NMTOKEN #REQUIRED
7            ref_film IDREF #REQUIRED>
8  <!ELEMENT film (titre, année>
9  <!ATTLIST film film_id ID #REQUIRED>
10           acteurs IDREFS #IMPLIED>
11 <!ELEMENT titre (#PCDATA) >
```

DTD: Pourquoi Validation?

Une DTD est une description de *l'interface* entre le *producteurs* et les *consommateurs* des données/documents XML :

- le producteur peut contrôler la qualité des données/documents produits
- le consommateur peut séparer la vérification syntaxique des données/documents (parseur) de la logique de l'application

ELEMENT : Déclaration du Type de Élément

Un élément est défini par un nom et un *modèle de contenu* :

- *Expression régulière* sur l'alphabet des noms d'éléments;
- *EMPTY* = élément vide;
- *ANY* = toute combinaison de tous les éléments;
- *#PCDATA* = texte
- Contenu mixte : `(#PCDATA | A | B ...)*`

Exemples

Un cinéma a

- un nom, une adresse optionnelle et
- une suite de séances.

```
<!ELEMENT cinéma (nom,adresse?,(séance)*)>
```

Une personne a

- un nom, plusieurs numéros de téléphone et
- au moins une adresse email

```
<!ELEMENT personne (nom,tel*,email+)>
```


ATTLIST : Déclaration des Attributs

```
<!ATTLIST élément nom type mode [defaut]>
```

Les éléments de type `séance` ont un attribut `heure` et un attribut `ref_film` :

```
<!ATTLIST séance heure NMTOKEN #REQUIRED
                ref_film IDREF #REQUIRED>
```

Les éléments de type `film` ont un attribut `film_id` et un attribut `acteurs` :

```
<!ATTLIST film film_id ID #REQUIRED
              acteurs IDREFS #IMPLIED>
```

Types d'attributs

- Chaînes de caractères : `CDATA`
- Énumérations : séquences de valeurs alternatives séparées par |
- ID, IDREF, IDREFS : identifiants et références
- ENTITY/ENTITIES : entité(s)
- NMTOKEN/NMTOKENS : chaîne(s) de caractères sans blancs
- NOTATION : notation

ATTLIST : Mode

Modes d'attributs:

- `#REQUIRED` : la valeur doit être définie
- `#IMPLIED` : la valeur est optionnelle
- `#FIXED` : la valeur est constante

```
<!ATTLIST séance heure NMTOKEN #REQUIRED
               ref_film IDREF #REQUIRED>
<!ATTLIST film film_id ID #REQUIRED
             acteurs IDREFS #IMPLIED
             langue (AN|FR|AL|ES|IT) #IMPLIED>
<!ATTLIST adresse ville CDATA #IMPLIED 'Paris'>
```


Résumé sur les DTDs

- Une DTD décrit la structure d'un ensemble de documents XML valides ;
- Tous les parseurs XML permettent de valider un document XML par rapport à une DTD ;
- Une DTD n'est pas un document XML ;
- Il existent des langages plus riches pour la description d'un document XML : XML Schema, Relax NG $\rightarrow$ grammaires d'arbres régulières