

XML et Programmation Internet

Cours 2

kn@lri.fr

Plan

1 Introduction, UTF-8 et XML ✓

2 XPath

2.1 Modèle d'arbre

2.2 XPath, introduction

XML vu comme un arbre (1/2)

- Tout ce qui apparaît dans le document correspond à un nœud de l'arbre (texte, balises, commentaires, blanc, ...)
- Il existe en plus, un nœud fictif se trouvant au dessus de l'élément racine, le **nœud document**
- Un couple balise ouvrante/balise fermante correspond à **un seul nœud**
- Les principaux types de nœuds sont: élément, attribut, texte, commentaire, document

XML vu comme un arbre (2/2)

Un document XML peut être vu comme un arbre:

```
<bibliography>
```

```
<book>
```

```
<title>Foundations of Databases</title>
```

```
<author>Abiteboul</author>
```

```
<author>Hull</author>
```

```
<author>Vianu</author>
```

```
<publisher>Addison Wesley</publisher>
```

```
<year>1995</year>
```

```
</book>
```

```
<book>
```

```
<title>The Lord of the Rings</title>
```

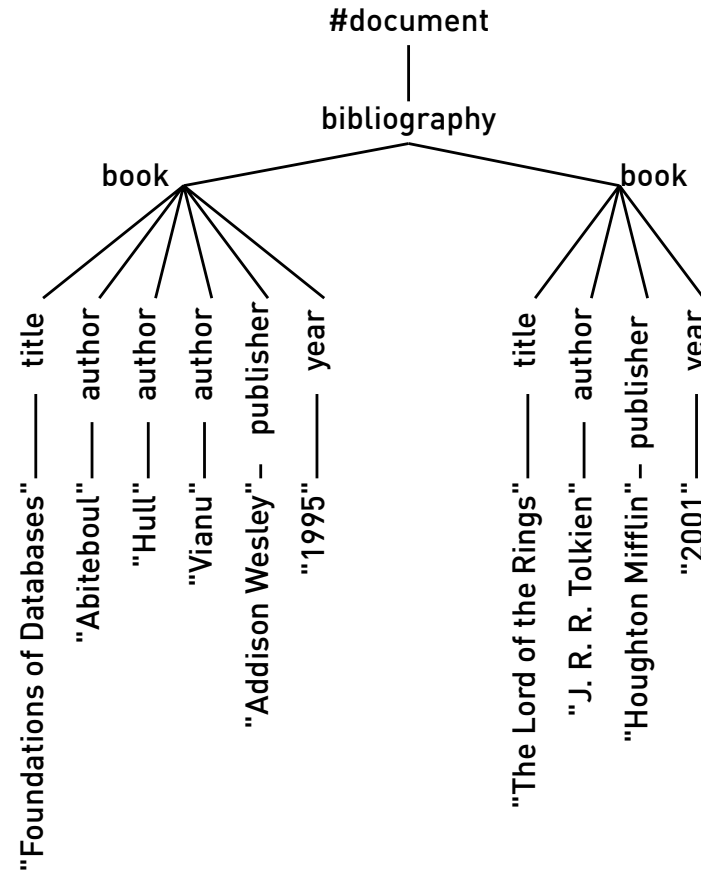
```
<author>J. R. R. Tolkien</author>
```

```
<publisher>Houghton Mifflin</publisher>
```

```
<year>2001</year>
```

```
</book>
```

```
</bibliography>
```



Sérialisation d'un arbre sous forme de document

Étant donné un arbre, comment peut on produire le document XML correspondant ?

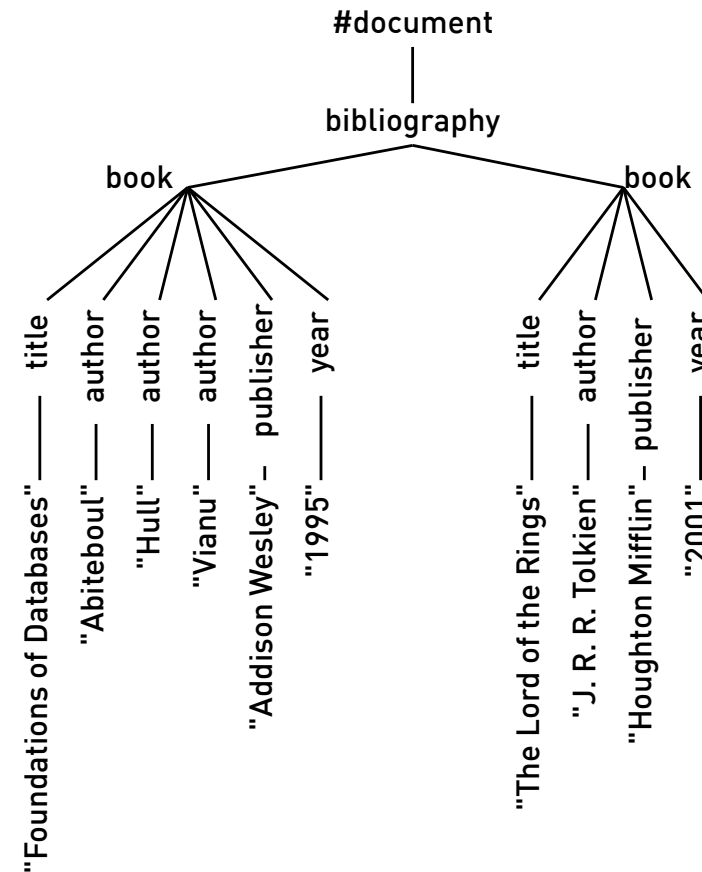
```
//pseudo-code
void print(Node n)
{
    if (n is text or comment) { output_text(n) }
    else {
        output_text ("<" + tag(n) + ">");
        for k in children(n)
            print(k);
        output_text ("</" + tag(n) + ">");
    }
}
```

- On effectue un parcours en profondeur d'abord
- Si le nœud courant est une feuille, on l'affiche
- Sinon on affiche la balise ouvrante, puis récursivement tous les fils, puis la balise fermante

Ordre du document, parcours préfixe

On appelle **ordre du document** un ordre **total** sur les nœuds d'un document qui correspond à leur ordre dans un fichier sérialisé. Il correspond aussi à la numérotation lors du parcours préfixe

1. #document
2. bibliography
3. book
4. title
5. "Foundations of Databases"
6. author
7. "Abiteboul"
8. author
9. "Hull"
10. author
11. "Vianu"



11. "Vianu"

Construction d'un arbre à partir d'un fichier XML ?

```
type Node = { label : string; children : List<Node> }
Stack<Node> stack;
stack.push (new Node("#document"), []));
while (true) {

    tag = read ();
    if end_of_file () break;
    if tag is opening {
        parent = stack.peek();
        node = new Node(tag, []);
        parent.addChild(node);
        stack.push(node);
    }
    if tag is closing {
        stack.pop();
    }
}
```

En pratique, on utilise des bibliothèques toutes faites pour lire/écrire des fichiers!

Plan

1 Introduction, UTF-8 et XML ✓

2 XPath

2.1 Modèle d'arbre ✓

2.2 XPath, introduction

Intérogation de documents XML

Les documents représentant des données (semi-) structurées, on souhaite en extraire de l'information

On va pouvoir écrire des requêtes sur des **critères scalaires** (« renvoyer tous les livres publiés après 2000 »), mais aussi sur des critères de **structure** (« renvoyer tous les éléments qui ont un fils author »)

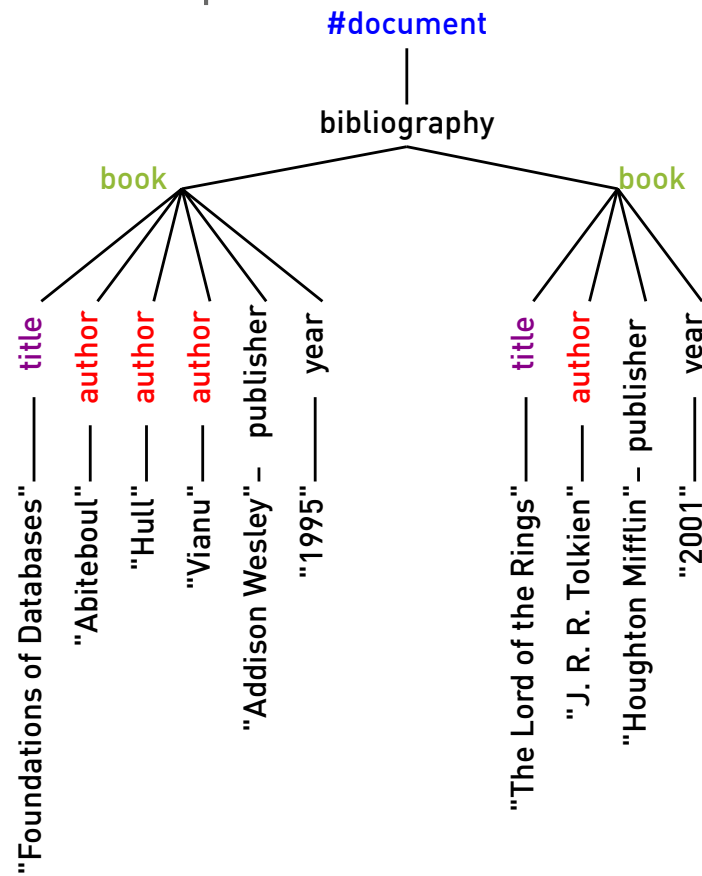
XPath

XPath est un langage de **selection de nœud** dans un document XML. Il ne permet **que** de sélectionner des nœuds, pas d'en construire de nouveaux. C'est un langage restreint qui ne contient pas de fonctions, variables, ... On peut le voir comme un équivalent du **SELECT de SQL**

XPath (exemple)

Sélectionner tous les titres du document (de manière compliquée)

`/descendant::author/parent::book/child::title`



XPath : syntaxe

La syntaxe d'une requête XPath est:

```
/axe1::test1[ pred1 ]/ ... /axen::testn[ predn ]
```

- **axe** : self, child, descendant, parent, ...
- **test** : node(), text(), *, ou un nom d'élément
- **pred(icat)** : chemin XPath, expression arithmétique, comparaison, ...

exemple:

```
/descendant::book[ child::year > 2000 ] / child::title
```

XPath : sémantique

Étant donné la requête:

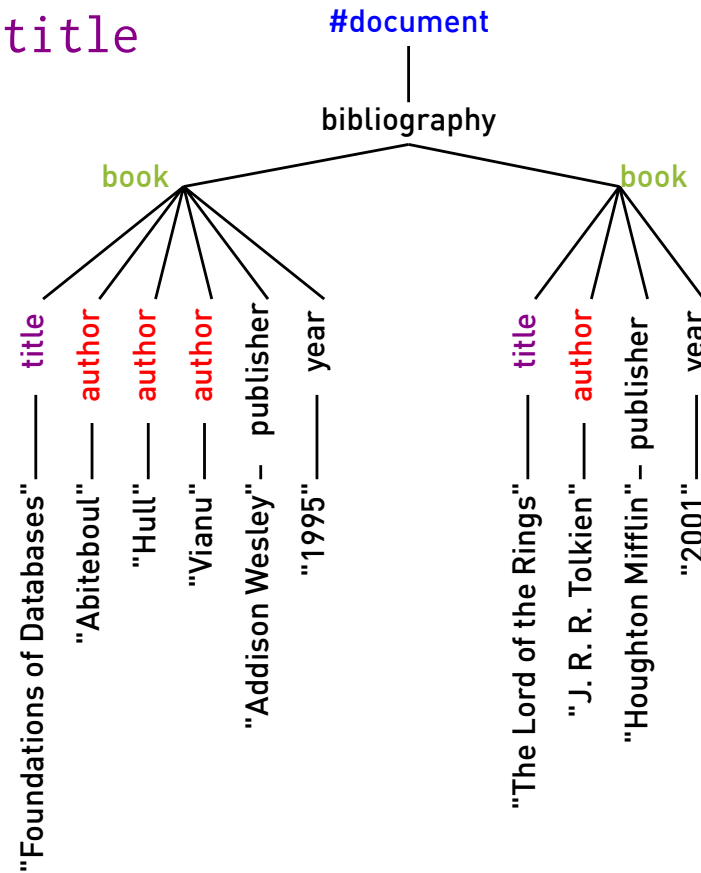
`/axe1::test1[pred1]/ ... /axen::testn[predn]`

1. le **nœud contexte** au nœud **document**
2. on sélectionne l'ensemble A_1 **tous les nœuds** qui sont dans l' axe_1 par rapport au nœud **contexte**
3. on sélectionne l'ensemble T_1 des nœud de A_1 qui vérifient le test $test_1$
4. on sélectionne l'ensemble P_1 des nœud de T_1 qui vérifient $pred_1$
5. On réapplique le pas 2 sur P_1
6. ...

XPath : sémantique (exemple)

`/descendant::author/parent::book/child::title`

1. On sélectionne le nœud document
2. On sélectionne tous les descendants
3. On filtre en ne gardant que les nœuds **author** ($T_1 \equiv P_1$)
4. Sur chacun des **author** on prend le parent (on n'obtient que 2 parents car on garde des ensembles de noeuds)
5. On filtre les parents pour ne garder que ceux qui sont **book**
6. On sélectionne tous les fils de chacun des **book**
7. On ne garde que les fils qui ont le tag **title**



XPath : axes

Le standard XPath définit un grand nombre d'axes

- **self** : on reste sur le nœud courant
- **child** : tous les fils du nœud courant
- **parent** : le parent du nœud courant. Seul le nœud document n'a pas de parent
- **descendant** : les fils, les fils des fils, etc. du nœud courant
- **ancestor** : le parent, et le parent du parent, etc. du nœud courant
- **descendant-or-self, ancestor-or-self** : comme les précédents mais inclus le nœud courant
- **following-sibling**: les frères se trouvant après
- **preceding-sibling**: les frères se trouvant avant
- **following, preceding, attributes** : usage avancé

XPath : les tests

On peut sélectionner des nœuds selon les critères suivants

- **node()** : n'importe quel nœud
- **text()** : un nœud texte ("The Lord of the Rings")
- ***** : n'importe quel élément (author, title, ...)
- **nom_d_element** tous les éléments ayant ce nom

XPath : prédicats (syntaxe)

```
p ::=  p or p
      |  p and p
      |  not (p)
      |  count(...), contains(...), position(), ...
      |  chemin XPath
      |  e1 op e2
```

e₁ et e₂ sont des expressions arithmétiques, op peut être <, >, =, !=, +, -, *, /, mod, ...

XPath : prédicats (sémantique)

On évalue le prédicat et on converti son résultat en valeur de vérité. Si la valeur vaut vrai, on garde le nœud courant, si elle vaut faux, on ne le garde pas

XPath connaît **4 types de données** pour les prédicats :

- Les booléens, valeur de vérité : vrai ou faux
- Les nombres (flottants), valeur de vérité compliquée...
- Les chaînes de caractères, chaîne vide = faux, sinon vrai
- Les ensembles de nœuds, ensemble vide = faux, sinon vrai

XPath : prédicats (exemples)

- `/descendant::book [child::title]` : sélectionne chaque élément book pour lequel l'ensemble des fils de nom title n'est pas vide
- `/descendant::book [count(child::author) > 2]` : sélectionne chaque book qui a plus de deux fils author
- `/descendant::book [contains(child::title, "Ring")]`
- `/descendant::book [count(child::author) > 2
or contains(child::author, "Tolk")
]/child::title`

Caractéristiques d'XPath

- XPath est un langage standardisé par le W3C
- Assez verbeux
- Langage de requêtes **monadique** (on ne peut renvoyer que des ensembles de nœuds. Par exemple il est impossible de renvoyer des ensembles de paires auteur/titre de livre)
- Il est assez compliqué à implémenter efficacement