Bases de données avancées

XML-DTD-XPath

Équipe pédagogique BD

http://liris.cnrs.fr/marc.plantevit/doku/doku.php?id=lifbdw2_2017a

Version du 24 novembre 2017

XML

eXtensible Markup Language

- Standard du W3C
- Objectif : stocker des données sous forme de texte
- Modèle de données : arbre (graphe)

Un ensemble de technologies

- Description de forme de documents :
  - DTD, Xml Schema, Relax NG
- Désignation de parties de documents :
  - XPath
- Liens inter/intra documents :
  - XLink (inter), XPointer (intra)
- Transformation de documents
  - XSLT
- Bases de données dédiées XML :
  - XQuery
- APIs de programmation (lecture/parcours/écriture)
  - DOM, SAX, StAX (Java)

Modèle de données en arbre

$\neq$ types de nœuds :

- racine (document) : possède exactement 1 enfant de type élément, qui va contenir les données
- élément
- attribut
- texte
- commentaire
- commandes (*processing instructions*)

DOM : modèle objet ayant pour but de représenter et de manipuler de tels arbres.

Syntaxe

- A base de balises, comme HTML
- Prologue (optionnel) : donne des informations pour la lecture du documents :
  `<?xml version="1.0" encoding="utf-8"?>`
- Déclaration de DTD (optionnel) : spécifie la forme du document :
  `<!DOCTYPE element-principal spec-DTD[ ... déclarations ...]>`
- Eléments et attributs :

```
<nom att1="val1" att2='val2'>
    Enfants
</nom>
```

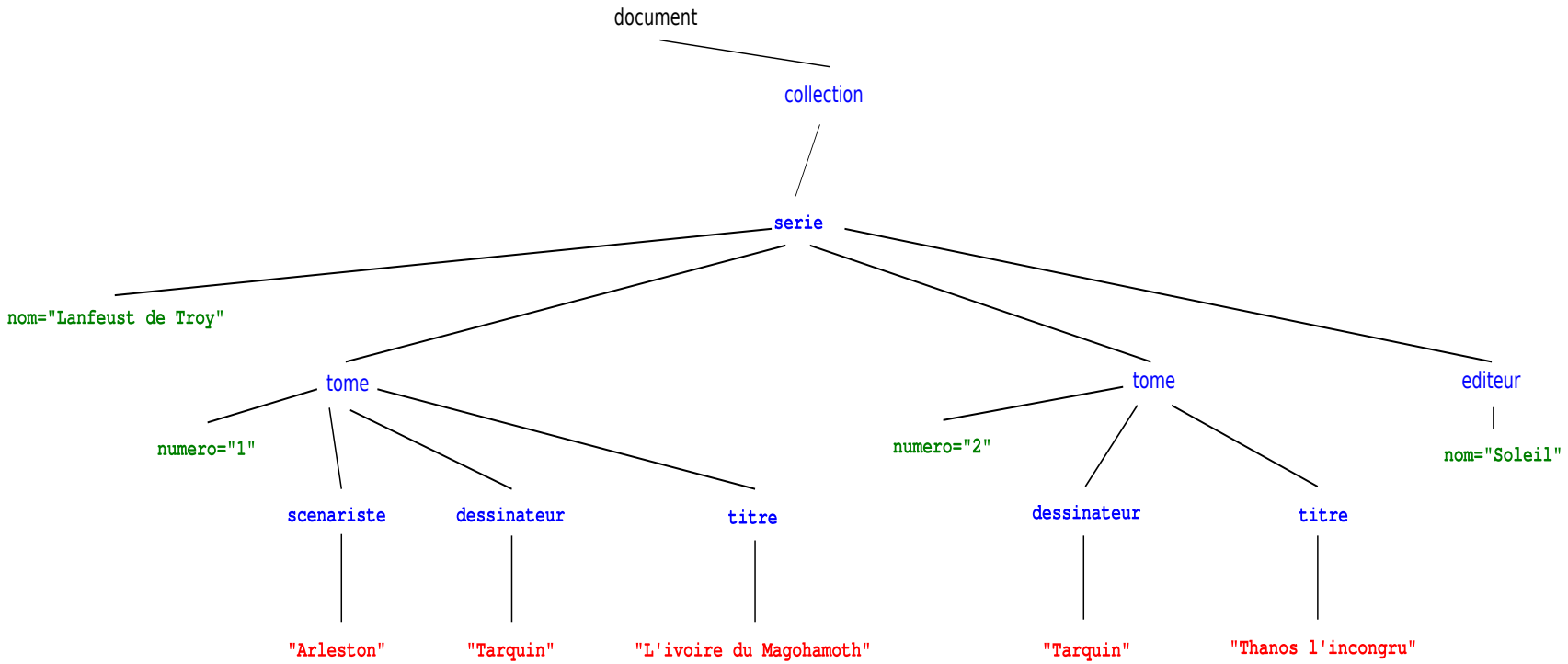
ou bien :

```
<nom att1="val1" att2='val2'/>
```

Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE collection SYSTEM "collecbd.dtd">
<collection>
  <serie nom="Lanfeust de Troy">
    <tome numero="1">
      <scenariste>Arleston</scenariste>
      <dessinateur>Tarquin</dessinateur>
      <titre>L&apos;ivoire du Magohamoth</titre>
    </tome>
    <tome numero="2">
      <dessinateur>Tarquin</dessinateur>
      <titre>Thanos l&apos;incongru</titre>
    </tome>
    <editeur nom="Soleil"/>
  </serie>
</collection>
```

Exemple - arbre



Namespaces

Schema

XPath

XQuery

Espaces de nommage

- Ambiguïté sur les noms XML
  - Problème similaire aux modules/packages en programmation
- Nom qualifié = Espace de nommage + nom local
- Espace de nommage : une URI
- Nom local : plus ou moins un identifiant dans un langage de programmation

  `([A-Z]|"_"|[a-z]|...)([A-Z]|"_"|[a-z]|"-"|"."|[0-9]|...)*`
- Syntaxiquement :
  - *nomLocal*
    utilise un espace de nommage par défaut
  - *prefixe*:*nomLocal*
    l'espace de nommage est celui rattaché à *prefixe*

Espaces de nommage : déclarations

- Via des attributs spéciaux,
  - valeur de l'attribut = espace de nommage concerné
- Portée : élément contenant l'attribut spécial et tous ses descendants
- Attribut `xmlns` : définit l'espace de nommage par défaut pour les éléments
- Attribut `xmlns:`*`prefixe`* : attache un espace de nommage au préfixe *`prefixe`*

Exemple

```
<?xml version="1.0" encoding="utf-8"?>
<livres xmlns="http://www.livres-pas-chers.com">
  <livre xmlns:encyclo="http://toutsurleslivres.org"
         ISBN="123456">
    <auteur encyclo:nat="Américain">
       Stephen King
    </auteur>
    <titre>Le fléau</titre>
    <annee>2003</annee>
    <encyclo:annee>1978</encyclo:annee>
    <prix>5.3</prix>
  </livre>
</livres>
```

noir : pas d'espace de nommage

Namespaces

Schema
- Introduction
- DTDs

XPath

XQuery

Qu'est qu'un schema ?

- Relationnel : Ensemble de contraintes que doit vérifier une instance d'une BD
  - Attributs des tuples d'une relation
  - Contraintes de type
  - Contraintes de clé
  - ...
- XML : Ensemble de contraintes structurelles que doit vérifier un document XML
  - Attributs/Enfants autorisés/requis dans un éléments
  - Type des valeurs pour les attributs et le texte
  - ...

$\Rightarrow$ DTD, XML Schema

Langages de schema pour XML

- Les plus connus :
  - **DTD** : Document Type Definition
    - Pas de gestion des espaces de nommage
  - XML Schema
    - Syntaxe XML qui peut prêter à confusion
  - Relax NG
- Certains schemas sont publics
  - Ex : XHTML, SVG, SOAP, MathML, OpenDocument, OpenXML, ...

DTD : Elements

<!ELEMENT *nom contenu*>

- Décrit les suites d'enfants possibles pour un élément.
- *contenu* peut être :
  - EMPTY : pas d'enfant
  - ANY : contenu arbitraire
  - (#PCDATA|nom_1|nom_2|...) : mélange de texte et d'éléments
  - (*expr*) : expression rationnelle de nom d'éléments

$$
\begin{array}{rll}
expr & ::= & expr_1\,,expr_2 \\
 & & expr* \\
 & & expr? \\
 & & expr+ \\
 & & expr_1\,|\,expr_2 \\
 & & nom \\
 & & (expr)
\end{array}
$$

DTD : Attributs

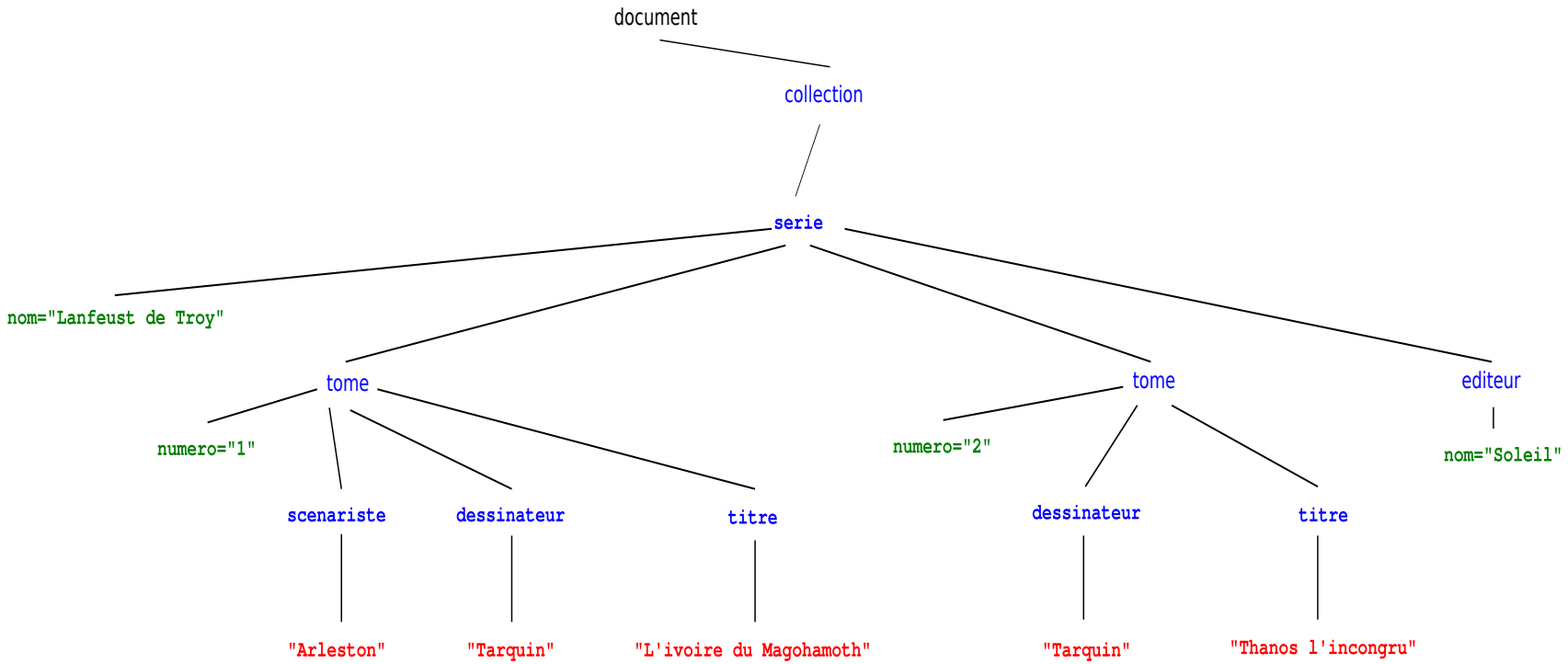
<!ATTLIST *nom* dec_1 dec_2>

- Décrit les attributs possibles pour un élément
- *dec* peut être :
  - *nom type*"*valeur*"
  - *nom type*#REQUIRED
  - *nom type*#IMPLIED
- *type* peut être :
  - CDATA, ID, IDREF, IDREFS
  - (*val*$_1$|*val*$_2$|...)

DTD : Exemple

```
<!ELEMENT collection (serie*)>
<!ELEMENT serie (tome+,editeur?)>
<!ATTLIST serie nom CDATA #REQUIRED>
<!ELEMENT tome (scenariste?,dessinateur?,titre)>
<!ATTLIST tome numero CDATA #REQUIRED>
<!ELEMENT scenariste (#PCDATA)>
<!ELEMENT dessinateur (#PCDATA)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT editeur EMPTY>
<!ATTLIST editeur nom CDATA #REQUIRED
                  adresse CDATA #IMPLIED>
```

Exemple - arbre



Namespaces

Schema

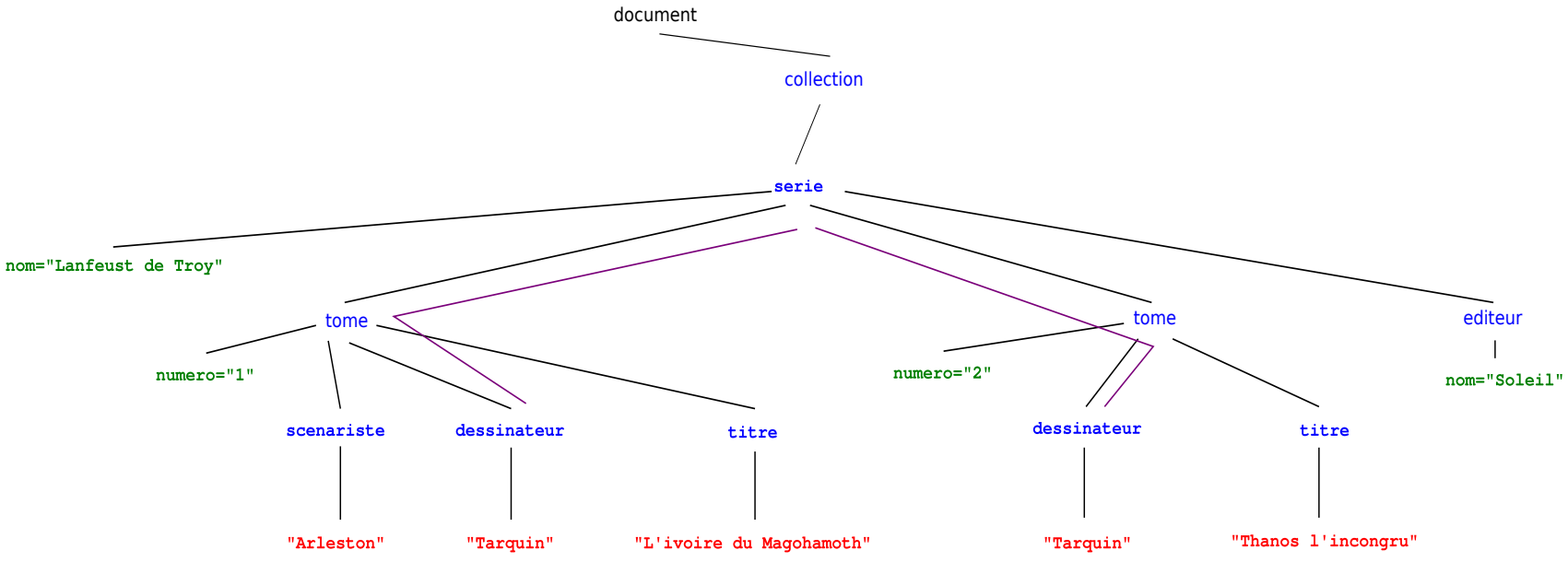
XPath

XQuery

XPath

- Objectif : sélection de morceaux de documents XML
- Utilisé dans d'autre langages
  - XQuery, XSLT, XPointer, WS-BPEL
  - Utilisable via des bibliothèques Java, Python, C, ...
- Principe : spécification de chemins dans l'arbre menant aux morceaux intéressants
  - expression XPath + noeud de départ
    $\rightarrow$ ensemble de chemins dans l'arbre XML
    $\rightarrow$ ensemble de noeuds sélectionnés

Exemple



"Aller sur un élément `tome`, puis sur un élément `dessinateur`"
Evaluer à partir de l'élément `série`

Valeurs

Types de valeur possible :

- suite de noeuds
- chaînes de caractères
  - conversion depuis un élément : concaténation de tous les noeuds texte descendants de l'élément
  - conversion depuis un attribut : valeur de l'attribut
  - conversion depuis une suite de noeud : concaténation
- nombres
  - conversion possible depuis une chaîne de caractères
- booléens
  - conversion implicite complexe (*c.f.* prédicats XPath)

Expressions de chemin

- Suite d'étapes séparée par "/"
- un "/" en début d'expression : départ forcé depuis la racine (document)
- Une étape est de la forme $axe::test[predicat]$
  - Le prédicat est optionnel
- Pour chaque étape, pour chaque noeud n d'ensemble N_d de noeuds de départs :
  - Calculer N_n^a obtenu en suivant l'axe à partir de n
  - Calculer N_n^t en filtrant N_n^a via le test
  - Calculer N_n^p en filtrant N_n^t via le predicat
- Résultat de l'évaluation de l'étape : $\bigcup_{n \in N_d} N_n^p$

Axes

forward	backward
child	parent
descendant	ancestor
attribute	
self	
descendant-or-self	ancestor-or-self
following-sibling	preceding-sibling
following	preceding

Tests & prédicats

Tests :

`element()`	`element(`*nom*`)`
`attribute()`	`attribute(`*nom*`)`
`*`	*nom*
`text()`	
`comment()`	
`processing-instruction()`	
`node()`	

Prédicats :

- Expression booléenne

Expressions booléennes

- Expressions classiques :
  - `and`, `or`, `not(...)`
  - fonctions renvoyant un booléen
- Nombre n : seul le n-ième élément de N_n^t est conservé
- Expression de chemin :
  - évaluation à partir du noeud à tester ;
  - vrai si résultat non vide
- Si une expression de chemin apparaît comme argument d'une fonction/d'un opérateur non booléen :
  - Evaluer l'expression à partir du noeud à tester ;
  - la (sous) expression booléenne est vraie si une des valeurs obtenues rend l'expression booléenne vraie

Abréviations

$$
\begin{aligned}
\texttt{child::}\textit{test} &\leftrightarrow \textit{test} \\
\texttt{attribute::}\textit{test} &\leftrightarrow \texttt{@}\textit{test} \\
\texttt{xxx/descendant-or-self::node()/yyy} &\leftrightarrow \texttt{xxx//yyy} \\
\texttt{parent::node()/xxx} &\leftrightarrow \texttt{../xxx} \\
\texttt{axe::test[(pr}_1\texttt{) and (pr}_2\texttt{)]} &\leftrightarrow \\
&\qquad \texttt{axe::test[pr}_1\texttt{][pr}_2\texttt{]}
\end{aligned}
$$

Exercice

Revoir la DTD “collection”.
Donner une expression XPath pour obtenir :

1. le premier tome de la collection dans chaque série
2. l’ensemble des titres d’album (sans la balise titre)
3. les séries dont on connaît l’éditeur
4. les séries dont on possède le tome numéro 1
5. le titre des albums dont le numéro est plus grand ou égal à 3

Expressions avancées : parenthèses

Parenthèses

- La partie *axe*::*test* peut être remplacée par une expression entre parenthèses
  - on peut appliquer un prédicat sur le résultat
  - important pour les prédicats type *n*-ième
- Exemple : le troisième tome de la collection : `/collection/(serie/tome)[3]`

Expressions avancées : fonctions

Fonctions prenant et ou renvoyant des ensembles de noeuds

- L'appel à la fonction est :
  - utilisé dans un prédicat
  - le point de départ d'un expression de chemin
    - remplace la première étape
    - voir l'expression complète
- Exemple : Les séries également présentes dans collection2.xml :
  `//serie[@nom=document('collection2.xml')//serie/@nom]`

Namespaces

Schema

XPath

XQuery

XQuery

- Langage de requête pour les documents XML
  - Utilisé en particulier dans les BD XML
- Fabrique des (morceaux de) documents XML à partir de documents XML
- Une expression XPath est une expression XQuery
- Permet de construire des morceaux de document : syntaxe XML + expressions XQuery entre accolades

Exemple

```
<personnes>
  <scenaristes>
    {//scenariste}
  </scenaristes>
  <dessinateurs>
    {//dessinateur}
  </dessinateurs>
</personnes>
```

FLWOR

for $v$$_1$ in e_1, $v$$_2$ in e_2, ...
let $w$$_1$:= e'_1, $w$$_2$:= e'_2, ...
where *condition*
order by eo_1, eo_2, ...return *expr*

- $v$$_i$, $w$$_i$: variables
- e_i, e'_i : expressions XPath
  - une variable peut remplacer la première étape d'un chemin
- eo_i : expression XPath (avec variables), suivie de `ascending` (par défaut) ou de `descending`
- *expr* : expression XQuery (contenant en général des constructions XML)

FLWOR : Evaluation

- Evaluer les combinaisons de valeurs possibles pour les $\$v_i$
  - On obtient un ensemble de tuples de valeurs
- Pour chaque tuple :
  - Evaluer les $\$w_j$
    - Si plusieurs valeurs pour une variable : elles sont concaténées
    - Les valeurs sont associées au tuple
- Filtrer les tuples avec la condition
- Pour chaque tuple, pris dans l'ordre de la clause order by, évaluer expr
  - Le résultat du FLWOR est la concaténation des résultats ainsi obtenus

Exemple

```
for $to in //tome
let $ti := $to/titre
where $to/@numero >= 3
order by $ti descending
return
<album>
  {$to/@numero}
  {$ti}
  <serie>{$to/../@nom}</serie>
</album>
```

Déclarations

- Précède l'expression (i.e. mettre au début du programme)
- `declare namespace` *`nomprefixe`*`="`*`uri_espace_nommage`*`";`
- `declare default element namespace "`*`uri_espace_nommage`*`";`
- `declare function` *`nomQualifie`* `(`*`$arg1`* `as` *`type1`*`, `*`$arg2`* `as` *`type2`*`, ...) as` *`type_retour`*
  `{` *`corps de la fonction`* `};`
- `declare default function namespace "`*`uri_espace_nommage`*`";`

Fin du cours.