XML, langage de structuration documentaire

Langage de transformation
et
langage d'interrogation

XSL, Xquery et Xpath : deux Langages, un sous Langage

- **Xpath** est un sous langage de navigation dans la structure et d'analyse des contenus exploité par XSL, XSLT et Xquery.

- **XSL** est un langage de spécification de transformation initialement dédié au formatage de la présentation (**XSL:FO**)
  - **XSLT**, un langage de spécification générique de toutes sortes de transformations à partir de ressources XML.

- **Xquery** est un langage de requête dans une structure XML.

Xpath : la navigation dans les arborescences XML

Un document XML se décrit comme un arbre, dont les éléments et les attributs sont des noeuds, sur lesquels Xpath « navigue ».

La première façon de décrire et de parcourir les noeuds de l'arbre d'un document XML est semblable à la description et à la navigation dans l'aborescence d'un système de fichier UNIX :

« **/** » désigne la racine ;

« **`/root/*/Name`** » désigne les éléments **Name** sous les premiers éléments appartenant à la racine **root** ;

« **`@`** »préfixe la désignation d'un noeud attribut ;

« **`/root/références//@Id`** » désigne les attributs **Id** de tous les éléments quelque part sous l'élément **références** situé sous la racine **root** ;

« **`.`** » désigne le noeud courant ;

« **`..`** » désigne le noeud parent.

Xpath : un langage de navigation explicite

Xpath apporte aussi des expressions explicites de navigation dans l'arbre, à partir de n'importe quel noeud, selon le principe syntaxique :

axisname::nodetest[predicate]

Exemple : **child::book[@price ="12€"] ,** sélectionne tous les éléments « **book** » sous l'élément courant dont l'attribut **price** est 12€

self : désigne le noeud courant ;
child : désigne les noeuds descendants directs du noeud courant ;
parent : désigne le noeud possédant le noeud courant ;
ancestors : désigne la lignée des noeuds possédant le noeud courant ;
ancestors-or-self : idem, y compris le noeud courant ;
descendant : désigne la lignée des noeuds sous le noeud courant ;
descendants-or-self : idem, y compris le noeud courant ;
attribute : désigne les noeuds attribut ;
following : désigne les noeuds suivant après le noeud courant du même parent;
following-sibling : idem pour le seul noeud suivant ;
preceding : désigne les noeuds précédent après le noeud courant du même parent;
preceding-sibling : idem pour le seul noeud précédent ;
namespace : désigne les noeuds attribut ;
text() : designe le texte d'un contenu

Xpath : les opérateurs

Xpath dispose d'opérateurs pour les recherches de noeuds et leur prédicats :

« | » indique le choix entre deux axes :
exemple : //to-be | //not-to-be

+ addition, **–** soustraction, ***** multiplication, **div** division;

= égal, **!=** non égal,

< inférieur, **<=** inférieur ou égal,

> supérieur, **>=** supérieur ou égal,

or ou, **and** et, **mod** modulo ;

Exemples :

```
//owl:ObjectProperty[rdfs:domain/owl:Class/@rdf:ID=$Name]
|//owl:ObjectProperty[rdfs:domain/owl:Class/@rdf:about=concat('#',$Name)]

//étape[position()=
floor(last()div2+0.5)orposition()=ceiling(last()div2+0.5)]
```

De Xpath 1.0 à 2.0, puis 3.0

- **Xpath 2.0** a apporté la possibilité d'appel de fonctions « utilisateur ».
- **Xpath 3.0** apporte la possibilité de définir des variables locales.
  - Ex : let $x := /*/@version return //e[@version = $x]
- Des alias de fonctions en ligne peuvent être déclarés, et utilisées comme arguments de fonctions d'ordre supérieur.

Des fonctions nouvelles standardisées, Algébriques et Booléennes.

- fn:abs(-23.4) renvoie 23.4
- fn:ceiling(23.4) renvoie 24
- fn:floor(23.4) renvoie 23
- fn:round(23.4) renvoie 23
- fn:round(23.5) renvoie 24
- fn:not(0) renvoie fn:true()
- fn:not(fn:true()) renvoie fn:false()
- fn:not("") renvoie fn:true()
- fn:not((1)) renvoie fn:false()

Fonctions standardisées Xpath (1/3)

- **Fonctions Algébriques**
  - fn:abs(-23.4) renvoie 23.4
  - fn:ceiling(23.4) renvoie 24
  - fn:floor(23.4) renvoie 23
  - fn:round(23.4) renvoie 23
  - fn:round(23.5) renvoie 24
- **Fonctions Booléennes**
  - fn:not(0) renvoie fn:true()
  - fn:not(fn:true()) renvoie fn:false()
  - fn:not("") renvoie fn:true()
  - fn:not((1)) renvoie fn:false()
- **Fonctions de traitement de chaînes de caractères**
  - fn:concat("HT","ML") renvoie "HTML"
  - fn:concat("HT","ML"," ","book") renvoie "HTML book"
  - fn:string-join(("HTML","book")," ") renvoie "HTML book"
  - fn:string-join(("1","2","3"),"+") renvoie "1+2+3"
  - fn:substring("HTML book",6) renvoie "book"
  - fn:substring("HTML book",3,5) renvoie "ML b"
  - fn:string-length("HTML book") renvoie 9
  - fn:upper-case("HTML book") renvoie "HTML BOOK"
  - fn:lower-case("HTML book") renvoie " html book"

Fonctions (2/3)

- Expressions régulières
  - fn:contains("HTML book","HTML") renvoie fn:true()
  - fn:matches("HTML book",« HTM..[a-z]*") renvoie fn:true()
  - fn:matches("HTML book",".*Z.*") renvoie fn:false()
  - fn:replace("HTML book","HTML","Web") renvoie "Web book"
  - fn:replace("HTML book","[a-z]","8") renvoie "HTML 8888«
- Evaluation de cardinalité
  - fn:exists(()) renvoie fn:false()
  - fn:exists((1,2,3,4)) renvoie fn:true()
  - fn:empty(()) renvoie fn:true()
  - fn:empty((1,2,3,4)) renvoie fn:false()
  - fn:count((1,2,3,4)) renvoie 4
  - fn:count(//rcp:recipe) renvoie 5

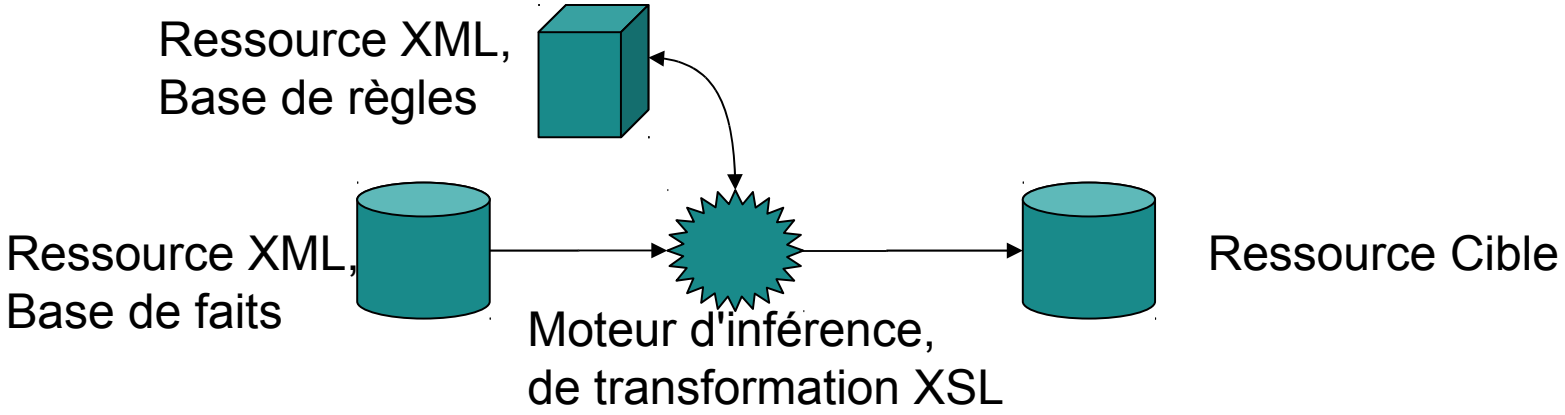
Fonctions (3/3)

- Transformations de séquences
  - fn:distinct-values((1, 2, 3, 4, 3, 2)) renvoie (1, 2, 3, 4)
  - fn:insert-before((2, 4, 6, 8), 2, (3, 5)) renvoie (2, 3, 5, 4, 6, 8)
  - fn:remove((2, 4, 6, 8), 3) renvoie (2, 4, 8)
  - fn:reverse((2, 4, 6, 8)) renvoie (8, 6, 4, 2)
  - fn:subsequence((2, 4, 6, 8, 10), 2) renvoie (4, 6, 8, 10)
  - fn:subsequence((2, 4, 6, 8, 10), 2, 3) renvoie (4, 6, 8)

- Xpath 3.0 apporte de nouvelles fonctions, dont : *head, tail, map, filter, map-pairs, pi, sin, cos, tan, asin, acos, atan, sqrt, format-integer,*

XSLT : De quoi s'agit il ?

- Un langage de spécification pour des transformations
  - Via des « feuilles de style » de règles (« Templates ») à appliquer,
  - Exécutées par des **moteurs d'inférence** dédiés : (Saxon, Xerces...)
- Un langage Initialement conçu pour la publication :
  - De XML en HTML,
  - De XML en PDF, via XSL-FO,
  - Puis de XML en n'importe quoi : XSLT,
- **XSL** existe en trois versions 1.0, 2.0, et depuis 2011 en 3.0



XSL, XSLT : les principes

Une feuille de style est un document XML,

dont la racine est l'élement « **`stylesheet`** » de l'espace de nom :

- http :// www.w3.org/1999/XSL/Transform

composé d'éléments «**`template`**», qui sont :

- Soit appliqués aux noeuds d'un document XML, découverts par une expression Xpath;
- Soit nommés pour être invoqués explicitement,

Les templates sont invoqués sans ordre, sauf préséance indiquée,

- Soit implicitement a priori,
- Soit explicitement par l'assertion :

```
<xsl:apply-templates/>
```

Le contenu d'un template effectue la composition du résultat à écrire : eléments XML, HTML ou autres...

Nota : XSLT est une généralisation de l'usage de XSL initialement destiné à XSL-FO, lui-même destiné au formattage des documents pour impression.

Exemple de feuille de style XSL

Présentation en tableau de la liste des liens vers les ingrédients d'une recette :

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html><body><h2>table des Ingrédients</h2>
    <table border="1">
      <tr bgcolor="#9acd32">
        <th>Ingrédient</th>
        <th>référence</th>
      </tr>
      <xsl:for-each select="//ingredient">
        <tr>
          <td><xsl:value-of select="."/></td>
          <td><xsl:element name="a">
                    <xsl:attribute name="href"><xsl:value-of select= '. /@ref'/></xsl:attribute>
                    <xsl:value-of select= '. /@ref'/>
              </xsl:element></td>
        </tr>
      </xsl:for-each>
    </table>
  </body></html>
</xsl:template>
</xsl:stylesheet>
```

Attachement de feuille de style à un document XML

Les navigateurs Internet intègrent (généralement) un moteur de transformation XSL.

Ils s'invoquent via une balise de traitement **<? ?>** :

Exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="recette.xsl"?>
<recette>
    <!-- (…) -->
</recette>
```

Cet attachement est ordinairement réalisé de façon programmatique, pour permettre différentes formes d'extractions d'un même document XML.

XSL 1.0, 2.0 et 3.0 , XPath 1.0, 2.0 et 3.0

- **XSL** :
  - 1.0 s'appuie sur la structure XML,
    - d'une entrée pour produire une sortie,
    - emploie des variables à assignation unique
  - 2.0
    - d'une entrée en produit plusieurs,
    - accepte des paramètres positionnels.
    - emploie des variables globales réassignables (pour saxonica)
  - 3.0
    - Implémente des fonctions de programmation supplémentaires.
- **XPath** ;
    - 1.0 navigue dans la structure,
    - 2.0 analyse les contenus, avec un jeu de fonctions,
    - 3.0 ajoute des fonctions nommées, assigne des variables.

De XSL 1.0 à 2.0, puis 3.0 (suite)

- **XSL 2.0 apporte les fonctionnalités suivantes :**
  - <xsl:value_of select=""/> accepte plusieurs valeurs,
  - <xsl:for-each traite n'importe quelle sorte de séquence, sans se limiter aux noeuds.
  - <xsl:perform-sort select="@rdfs:label" />renvoie une liste triée d'objet selon l'attribut choisi (par défaut l'identifiant)
  - La production de plusieurs sorties <xsl:result-document href="" >…</xsl:result-document>
  - L'appel directs à des templates nommés, <xsl:call-template name="monTemplate"> mutualisation de templates entre feuilles de style.
  - Le paramétrage d'appel des feuilles de style.
  - L'utilisation de Xpath 2.0.
- **XSL 3.0 apporte en plus** :
  - xsl:evaluate effectue l'évaluation d'une fonction Xpath,
  - xsl:try and xsl:catch permettent d'intercepter des erreurs,
  - xsl:iterate est une alternative à xsl:for-each qui autorise les interruptions par xsl:break.
  - …
- **Saxonica** propose des fonctionnalités supplémentaires anticipées du standard.
  - <saxon:assign autorise la réaffectation de valeurs à une variable

Le langage de requête Xquery

- **Xquery** est à XML ce qu'est SQL pour une base de données relationelles,
- Un processeur Xquery travaille :
  - Sur des documents XML,
  - Sur des bases de données XML,
  - « en mémoire », sur un arbre DOM.
- **Xquery** :
  - Permet de faire des requêtes selon la structure ou encore les contenus en se basant sur des expressions Xpath (version 2.0).
  - Peut générer des nouveaux documents,
    - autrement dit: on peut manipuler un résultat obtenu et y ajouter
  - N'effectue pas de mises à jour.
    - (pas d'équivalent de update/insert de SQL),
- **XQuery** permet d'extraire des fragments XML, d'y effectuer des recherches et de générer des fragments XML.

Usage de Xpath par Xquery

- Xquery fait appel à Xpath pour naviguer dans une structure XML
- Exemple de requète :
  - Rechercher les noeuds contenant <toto><lulu/></toto> :

```
for $t in //toto/lulu
   return $t
```

- Autre Exemple de requète :
  - Chercher tous les noeuds avec le chemin //toto/lulu dans la ressource "exemple.xml", publié sur le site democrite.org

```
for $t in
   document("http://democrite.org/exemple.xml")//topic/title
   return $t
```

Résultat de requète

- Soit la ressource « exemple.xml » :

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
  <lulu> contenu sans intérêt> </lulu>
  <toto>
      <lulu>un contenu intéressant> </lulu>
  </toto>
  <machin>
      <toto>
        <lulu> un autre contenu intéressant> </lulu>
      </toto>
  </machin>
</Root>
```

- La requète précédente donnerait la collection :

```
<lulu>un contenu intéressant> </lulu>
<lulu> un autre contenu intéressant> </lulu>
```

Expressions : For Let Where Order Return (1/3)

- **A. "for"**

"for" associe à chaque $variable une valeur (fragment XML) trouvé pour l'expression Xquery :

```
for $variable in expression_recherche
return
```

- **B. "let"**

"let" permet de d'assigner une valeur à une variable :

```
for $t in document("truc.xml")//machin//chose
let $MonRésultat := $t//chouette
return $MonRésultat
```

- **Autre exemple :**

```
for $t in document("doc.xml")//chapeaux
let $n := count($t//feutre)
return <result> {$t/title/text()} possède {$n} feutres </result>
```

Expressions : For Let Where Order Return (2/3)

- **C. "where"**

  where définit une condition de sélection :

  exemple n'affichant que les cas où on est en présence d'au moins 2 <feutre> :

```
for $t in document("doc.xml")//chapeaux
let $n := count($t/feutre)
where ($n > 1)
return <result> {$t/title/text()} possède {$n} feutres </result>
```

- **D. order**

  order effectue le tri les résultats :

```
for $t in document("doc.xml")//chapeaux
let $n := count($t//feutre)
where ($n > 1)
order by $n
return <result> {$t/title/text()} possède {$n} feutres </result>
```

Expressions : For Let Where Order Return (3/3)

- **E. return**

  * construit l'expression à retourner à chaque iteration,

  * Chaque iteration ne peut retourner qu'un seul fragment XML (pas une collection).

```
for $t indocument("doc.xml")//chapeaux
let $n := count($t//feutre)
return <result>
            {$t/title/text()} possède {$n} feutres
       </result>
```

Questions d'implémentations

- Il existe deux modes de traitement de ressources XML, dits « DOM » et « SAX » :
  - DOM signifie « Document Object Model »,
    - Un arbre DOM est une ressource XML représentée en mémoire,
    - Après avoir parsé un document, DOM permet de naviguer dans le document XML, de requérir le contenu des éléments et des attributs,
  - SAX signifie « Simple API for XML »,
    - Sax fonctionne de façon événementielle : toute ouverture ou fermeture d'un élément par l'analyse syntaxique déclenche une action.
    - SAX parcourt le document XML dans l'ordre du document et envoie les événements au fur et à mesure, sans mettre le document en mémoire.

Traitements DOM

- **Exemple en Python : chargement :**

```
from xml.dom.minidom import parse
  import sys
  xmlfilename = sys.argv[1]
  dom         = parse(xmlfilename)
```

- À partir de l'objet DOM, il est possible d'obtenir : l'élément racine du document XML à l'aide de **documentElement**, ou le noeud élément portant un nom particulier à l'aide de **getElementsByTagName**.
- À partir d' un noeud, il est possible de naviguer vers d'autres noeuds via des méthodes de parcours de chemins :
  - **childNodes** : la liste des noeuds fils ;
  - **parentNode** : le père ;
  - **firstChild** : le premier fils ;
  - **lastChild** : le dernier fils ;
  - **nextSibling** : le frère droit ;
  - **previousSibling** : le frère gauche.
- Si un noeud demandé n'existe pas, on obtient en Python la valeur **None**. Si un noeud n'a pas de fils, **childNodes** renvoie logiquement une liste vide.
- Les attributs d'un élément sont accessibles grâce à la méthode **attributes**

Exemple DOM

- Accéder au premier des fils de l'élément « villes » :

```
from xml.dom.minidom import parse
  import sys
  xmlfilename = sys.argv[1]
  dom         = parse(xmlfilename)
  villes  = dom.getElementsByTagName("ville")
  premier = villes[0]
  texte   = premier.childNodes[0]
  print texte.nodeValue
```

- Accéder aux attributs URL d'un ensemble d'éléments « site »:

```
from xml.dom.minidom import parse
  import sys
  xmlfilename = sys.argv[1]
  dom         = parse(xmlfilename)
  sites = dom.getElementsByTagName('site')
  for site in sites :
      attrs   = site.attributes
      urlnode = attrs['url']
      print urlnode.nodeValue
```

Edition DOM

- Création d'un arbre DOM :

```
impl    = getDOMImplementation()

newdoc   = impl.createDocument(None,"laracine",None)
  top_element = newdoc.documentElement
```

- Création d'un élément avec un attribut :

```
new_element = newdoc.createElement('MonElement')
  new_element.setAttribute('MonAttribut','MaValeur')
```

- Placer l'élément au sommet de la structure :

```
top_element.appendChild(new_element)
```

- Sérialiser l'arbre sous la forme d'un document XML :

```
PrettyPrint(newdoc)
```

Traitements SAX

- Création d'un Parser (en Python) :

```
import sys
   from xml.sax import ContentHandler,make_parser
   # création du parser
   parser = make_parser()
   # instanciation du handler
   handler = MyContentHandler()
   # affectation du ContentHandler
   parser.setContentHandler(handler)
   # traitement du fichier passé en paramètre
   parser.parse(sys.argv[1])
```

- SAX se compose de quatre interfaces appelées par le parser pour les différents événements rencontrés durant la lecture.
  Ces interfaces sont :
  - **ContentHandler**
  - **ErrorHandler**
  - **DTDHandler**
  - **EntityResolver**

Interfaces SAX (1/2)

- Définition et affectation d'un **ContentHandler** :

```
class MyContentHandler(ContentHandler) :
      def startElement(self, name, attrs) :
         print "Start element:", name
  handler = MyContentHandler()
  parser.setContentHandler(handler)
```

- Cet interface dispose (entre autres) des méthodes principales
  - **startDocument**: appelée au début du document.
  - **endDocument**: appelée à la fin du document.
  - **startElement**: appelée à chaque fois qu'une balise ouvrante est rencontrée.
  - **endElement**: appelée à chaque fois qu'une balise fermante est rencontrée.
  - **characters**: appelée à chaque fois qu'un noeud texte est rencontré., (le contenu texte d'un même élément peut être séparé en plusieurs parties).
  - **processingInstruction**: appelée pour chaque "processing instruction" rencontrée.

Interfaces SAX (2/2)

- **ErrorHandler**
  - L'interface ErrorHandler sert à gérer les erreurs rencontrées lors de la lecture. Elle sert par exemple à intercepter des erreurs dues à un document XML mal formé que l'on voudrait ignorer.

- **DTDHandler**
  - Cette interface sert à résoudre les notations et les entités définies dans la DTD associée au document XML parsé.

- **EntityResolver**
  - Cet interface sert à résoudre les entités externes contenues dans le document XML. Quand le parser rencontre une entité externe, il appelle la méthode **resolveEntity** écrite par le développeur.

Fin du module