Format de données du Web
Partie XML

Jérôme David
Université Grenoble Alpes
2015-2016

Origines

- XML = Extensible Markup Language
  - (Méta-)Langage pour la structuration de données sur le Web
  - Créé en 1996 par le « XML Working Group » sous égide du W3C
- W3C
  - World Wide Web Consortium est une communauté internationale de standardisation sur le Web
  - Elle développe des standards ouverts
  - Exemples de standards : HTML, CSS, PNG, XML
  - Fondé par le père du web : Tim Berners-Lee

XML, un Méta-langage à balises

- Un langage à balise est une classe de langages permettant de structurer/annoter un texte
  - Exemples : HTML, LaTeX
- Méta-langage : langage permettant de créer d'autres langages
- XML
  - décrit le format des balises et leur structuration (arborescente)
  - mais ne contient pas de balises prédéfinies
    - On peut créer ses propres balises !

Quelques objectifs

- Utilisable sur internet
- Etre générique
- Facile à être traité par des programmes
- Facile à être lu par des humains
- Les documents XML doivent être faciles à créer

En résumé, XML devait être un langage générique, portable et ouvert

```
<?xml version="1.0" encoding="UTF-8"?>
<外语>Китайська мова</外语>
```

Aperçu (rapide) de technos XML

- Quels sont les technos et langages XML que vous connaissez ?

Exemple de document XML

Quels sont les éléments structurant ce document ?

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <!-- Commentaire -->
  <ex:collection
    xml:lang="fr"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ex="http://exemple.org"
  >
    <élément>Texte</élément>
    <dc:title>Astérix le Gaulois</dc:title>
    <ex:livre attribut="valeur" type="BD">
      <dc:title>Astérix chez les Belges</dc:title>
      <!-- élément répété -->
      <dc:creator>René Goscinny</dc:creator>
      <dc:creator>Albert Uderzo</dc:creator>
      <dc:description>
        <b>Astérix chez les Belges</b> est un album de
        <a href="http://fr.wikipedia.org/wiki/Bande_dessinée">bande dessinée</a>
        de la série Astérix le Gaulois créée par René Goscinny et Albert Uderzo.
        <br /><!-- élément vide -->
        Cet album publié en 1979 est le dernier de la série écrit par René Goscinny.
      </dc:description>
    </ex:livre>
  </ex:collection>
```

Source : http://fr.wikipedia.org/wiki/Extensible_Markup_Language

Eléments

- Un élément est une pièce d'information
  - Commençant par une balise ouvrante

    ```
    <LeNomDeMaBalise>
    ```
  - Contenant ensuite du texte et/ou d'autres éléments
  - Terminant par une balise fermante

    ```
    </LeNomDeMaBalise>
    ```
- Exception : l'élément vide

  ```
  <BaliseVide/>
  ```

Attributs

- Permettent de préciser un élément
  - Placés dans la balise d'ouverture d'un élément
  - Paire clé – valeur
  - La valeur est entre guillemets (") ou quotes (')
    - On ne mélange pas les deux pour un même attribut
  - Plusieurs attributs possible
  - Ordre sans importance
  - Un nom d'attribut (clé) ne peut être utilisée qu'une fois pour un élément donné

```
<adresse type="travail" pays="france">
    1251, avenue centrale
    Campus Universitaire
    38400 Saint Martin d'Hères
</adresse>
```

Syntaxe des nom d'éléments et attributs

- Syntaxe d'un nom d'élément
  - Peut commencer par
    - une lettre (accentuée ou non),
    - Tiret bas _
    - Deux-points :
  - Ensuite on peut utiliser les chiffres, - et .
- Les nom sont sensible à la casse
  - « NomElément » est différent de « nomélément »
- Un nom d'élément ne doit pas commencer par xml (Xml, XML, etc.)

Référence d'entités

- Comment faire quand on veut utiliser les caractères protégés ?
  - Tels que <, &

<	<
>	>
&	&
'	'
"	''

Le prologue

- Tout document XML commence par une déclaration du type

```
<?xml version="1.0" encoding="UTF-8"?>
```

- Attributs possibles
  - version (obligatoire) : version du standard XML utilisé
  - encoding : Encodage des caractères utilisé ("UTF-8", "ISO-8859-1" , etc.)
  - Standalone : "no" si le document fait référence à des schémas de définition externes. Mettre "yes" sinon

Les autres types de noeuds

- Commentaires

```
< !-- un commentaire -->
```

- Instructions de traitement
  - Format : <?cible texte_a_interpreter ?>

```
<?xml-stylesheet href="default.css" title="Default style"?>
```

```
<?php
        for ($i=1;$i<10; $i++) {echo($i); }
?>
```

- Les sections d'échappement CDATA
  - Permet de délimiter du texte qui ne sera pas interprété

```
<![CDATA[
Je peux mettre ce que je veux
et même des <balises>
]]>
```

Notion de document XML

- Un document XML bien-formé est un texte
  - Respectant les contraintes syntaxiques énoncées auparavant
  - Commençant par la déclaration XML
  - Contenant **un et un seul élément racine** (qui peut contenir d'autres éléments)
    - Un document XML est un arbre : une racine et pas d'éléments entrelacés (i.e. qui se chevauchent)
- Pour plus de détails, voir la recommandation :
  - http://www.w3.org/TR/xml/

Exemples

- Cherchez les erreurs !

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Annuaire d'illustration -->
<annuaire>
     <personne <!-- une 1ere personne --> >
          <nom>Dupont</nom>
          <prénom>Paul <!-- ou Polo --> </prenom>
          <date format="ISO" format="fr-fr">1998-07-23
          <telephone/>
     </personne>
     <personne>
          <!-- une 2nd personne -->
          <nom>Alain</nom>
          <prenom courant='Vrai'>Adrien</prenom>
          <date format="ISO">1969-06-04</date>
          <telephone>
               <indicatif_tel> 02 </indicatif_tel>
               <num#tel> 40 40 36 18 </num#tel>
          </telephone>
     </personne>
</annuaire>
<annuaire>
     <personne>
        <nom>DUPONT</nom>
        <prenom>MARC</prenom>
        <telephone/>
     </personne>
</annuaire>
```

Solution

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Annuaire d'illustration -->
<annuaire>
     <personne <!-- une 1ere personne --> >
          <nom>Dupont</nom>
          <prénom>Paul <!-- ou Polo --> </prénom>
          <date format="ISO" format="fr-fr">1998-07-23</date>
          <telephone/>
     </personne>
     <personne>
          <!-- une 2nd personne -->
          <nom>Alain</nom>
          <prenom courant='Vrai'>Adrien</prenom>
          <date format="ISO">1969-06-04</date>
          <telephone>
               <indicatif_tel> 02 </indicatif_tel>
               <num#tel> 40 40 36 18 </num#tel>
          </telephone>
     </personne>
</annuaire>
<annuaire>
     <personne>
     <nom>DUPONT</nom>
     <prenom>MARC</prenom>
     <telephone/>
     </personne>
</annuaire>
```

Xmllint est un outil bien pratique permettant entre autre de vérifier la syntaxe XML

Les espaces de noms

- Un document XML peut avoir recours à des balises définies dans des vocabulaires XML différents
  - Comment faire quand il y a conflit de noms?
- La Solution : XML namespace
  - Un namespace est permet d'identifier un vocabulaire XML via une URI
    - URI = Uniform Resource Identifier (exemple : les URL)
    - Vocabulaire XML : ensemble de balises prédéfinies
  - Syntaxe : `<unPrefixe:balise xmlns:unPrefixe='http://c.org'>`
  - La visibilité du namespace est limitée au sous-arbre sur lequel l'attribut xmlns est déclaré

Exemple

- Comment distinguer les 2 balises « title » ?
  - Une pour la civilité
  - Une issue de la norme HTML 4

```
<directory>
    <people>
       <title>M.</title>
       <lastname>DUPONT</lastname>
       <firstname>MARC</firstname>
       <webpage_content>
        <html>
            <title>Marc's Webpage</title>
            <p>I am a cray XML designer working for company X</p>
        </html>
       </webpage_content>
    </people>
</directory>
```

Solutions

```
<d:directory xmlns:d="http://mycompany.org/directory">
     <d:people>
        <d:title >M.</d:title>
        <d:lastname>DUPONT</d:lastname>
        <d:firstname>MARC</d:firstname>
        <d:webpage_content>
          <h:html >
               <h:title>Marc's Webpage</h:title>
               <h:p>I am a cray XML designer working for company X</h:p>
          </h:html>
        </d:webpage_content>
     </d:people>
</d:directory>
```

```
<d:directory xmlns:d="http://mycompany.org/directory"
               xmlns:h="http://www.w3.org/TR/html4/">
     <d:people>
        <d:title >M.</d:title>
        <d:lastname>DUPONT</d:lastname>
        <d:firstname>MARC</d:firstname>
        <d:webpage_content>
          <h:html>
               <h:title>Marc's Webpage</h:title>
               <h:p>I am a cray XML designer working for company X</h:p>
          </h:html>
        </d:webpage_content>
     </d:people>
</d:directory>
```

Espace de noms par défaut

- On peut définir un espace de noms par défaut :

```
<directory xmlns="http://mycompany.org/directory"
               xmlns:h="http://www.w3.org/TR/html4/">
     <people>
        <title >M.</title>
        <lastname>DUPONT</lastname>
        <firstname>MARC</firstname>
        <webpage_content>
          <h:html>
               <h:title>Marc's Webpage</h:title>
               <h:p>I am a cray XML designer working for company X</h:p>
          </h:html>
        </webpage_content>
     </people>
</directory>
```

Exercice

Ecrivez un document XML représentant votre CV (utilisez au moins une fois des attributs)

Parsez-le avec xmllint pour vérifier qu'il est bien-formé

Validation XML : les schémas

- Document XML bien-formé =
  - Document respectant la syntaxe XML
  - Voir slides précédents
- Document XML valide
  - Doit être un document XML bien formé
  - Et doit se conformer à un schéma spécifié
    - Les langages de définition de schéma (ou de type de document)
      - DTD : Document Type Definition (W3C)
      - XML Schema (W3C)
      - Relax-NG (ISO)
    - Ces langages sont des grammaires de documents XML
    - Un schéma permet de décrire une classe de documents XML

Document Type Definition

- Une DTD est déclaré via la directive DOCTYPE
- Une DTD peut être interne ou externe
  - Exemple de déclaration externe :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Exemple</title>
  </head>
  <body>
    <p>Le <a href="http://upmf-grenoble.fr">site web</a> de l'UPMF.</p>
  </body>
</html>
```

Document Type Definition

- Exemple de déclaration interne

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE liste_de_gens [
 <!ELEMENT liste_de_gens (personne)*>
 <!ELEMENT personne (nom, date_de_naissance?, genre?, numero_de_secu?)>
 <!ELEMENT nom (#PCDATA)>
 <!ELEMENT date_de_naissance (#PCDATA)>
 <!ATTLIST personne genre (CDATA | masculin | féminin) "féminin">
 <!ELEMENT numero_de_secu (#PCDATA)>
]>
<liste_de_gens>
  <personne genre='masculin'>
    <nom>Fred Bloggs</nom>
    <date_de_naissance>2008-11-27</date_de_naissance>
  </personne>
</liste_de_gens>
```

Source : http://fr.wikipedia.org/wiki/Document_Type_Definition

XML DTD

- Pour les DTD, un document XML est fait de
  - D'éléments
  - D'attributs
  - D'entités
  - De texte (parsé : PCDATA, ou non : CDATA)

XML DTD - Eléments

- 2 façon de déclarer un élement
  - <!ELEMENT nom-de-l-element type>
    - type = EMPTY (élément vide) ou ANY (n'importe quelle donnée parseasble)
  - <!ELEMENT nom-de-l-element (contenu-de-l-elem)>
    - Contenu-de-l-elem peut être
      - #PCDATA : Texte qui sera parsé
      - AutreElement1 : un élément fils
      - Une combinaison
        - Liste séquentielle : element1, element2
        - Liste optionnelle : element1 | element2 | #PCDATA
        - **Note : une balise à contenu mixte (mélange de #PCDATA et d'éléments) ne peut être déclarée que sous forme de liste optionnelle**
    - Les quantifieurs : permettant de définir des cardinalités

      \+ : au moins 1

      * : zero ou plus

      ? : 0 ou 1

```
<!ELEMENT html (head, body)>
<!ELEMENT p (#PCDATA | p | ul | dl | table | h1 | h2 | h3)*>
```

XML DTD - Attributs

- Syntaxe de la déclaration d'attributs :

```
<!ATTLIST nom-element nom-attribut type valeur-attribut>
```

Type	***Description***
CDATA	Chaîne de caractères
(v1|v2|...)	Soit v1, soit v2, ...
ID	La valeur est un identifiant (unique)
IDREF	La valeur est un identifiant d'un autre élément
IDREFS	La valeur est une liste d'identifiants d'autre éléments
NMTOKEN	Nom XML valide mais sans restriction sur 1er caractère
NMTOKENS	Liste de noms XML valides
ENTITY	Une entité
ENTITIES	Une liste d'entités
NOTATION	Nom d'une notation
Xml :	Valeur XML prédéfinie

Valeur	***Description***
une-valeur	*une-valeur* est la valeur par défaut
#REQUIRED	Attribut obligatoire
#IMPLIED	Attribut optionnel
#FIXED *une-valeur*	La valeur est fixée à *une-valeur*

Exercice

En vous appuyant sur la DTD d'XHTML strict
http://www.w3.org/TR/2000/REC-xhtml1-20000126/DTD/xhtml1-strict.dtd

Créez une page HTML contenant une table qui utilise tous les éléments possible déclarés dans cette DTD.

Validez ce document

Vous pouvez utilisez xmllint ou un validateur en ligne

Exercice

Créez une DTD pour des types de documents XML représentant un CV (en fonction du CV créé précédemment).

Validez le CV créé lors de l'exercice précédent.

XML DTD – Entités

- Un moyen de définir des constantes en XML
  - Syntaxe : `<!ENTITY nom-entite "valeur">`
  - Utilisation dans un document XML : `&nom-entite ;`
- Exemple

```
<?xml version="1.0" standalone="yes" ?>
<!DOCTYPE copyright [
  <!ELEMENT copyright (#PCDATA)>
  <!ENTITY upmf "Université Pierre Mendès France">
]>
<copyright>&upmf;</copyright>
```

On peut importer des entité définies dans des DTD externe :

```
<!ENTITY nom-entite-a-importer SYSTEM 'url-de-la-dtd'>
```

XML Schéma

- XML Schema (XSD) est, comme les DTD, un langage de description de format XML
  - Recommandation du W3C en 2001
  - XML Schema est lui-même un langage XML
  - XML Schema apporte en plus des DTD :
    - La gestion des espaces de nom (namespaces)
    - des types de données prédéfinis (data, entiers, etc)
    - La possibilité de créer de nouveaux types
    - Définition de contraintes (ex : cardinalités arbitraires > 1)
  - Mais il ne gère pas les entités

Structure de base

On utilise l'espace de noms d'XML Schema

```
<?xml version="1.0" encoding="UTF-8" ?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://ic2a.upmf-grenoble.fr/ns"
            xmlns="http://ic2a.upmf-grenoble.fr/ns"
            elementFormDefault="qualified">

<!-- Déclaration des types, élements et attributs  -->

</xsd:schema>
```

Même les sous-éléments déclarés dans ce schéma seront associé à l'espace de noms spécifié auparavant

Ce schéma est associé à cet espace de noms. Ainsi tous les éléments et types déclarés dans ce schéma sont associés à ce namespace

Constituants de XSD

- Un schéma XSD est constitué de
  - Déclaration d'éléments et attributs
    - les types des éléments ou attributs
    - D'autres contraintes (cardinalité, etc)
  - Définition de types
    - Les types simples
      - N'ayant que du texte (ou rien)
      - XML schéma contient des types simples prédéfinis
        - Integer, double, boolean, time, string, etc.
        - ID, IDREF, ENTITY, NMTOKEN → issus des DTD
        - Liste : http://www.w3.org/TR/xmlschema-0/#CreatDt
    - Les types complexes
      - Types non atomiques composés d'éléments et d'attributs
      - Utilisés pour définir des types d'éléments

Définition de types simples

- Différents constructeurs :
  - Liste / union
  - Restrictions :

```
<xs:simpleType name="telephone">
  <xs:list itemType="xs:integer" />
</xs:simpleType>
```

```
<xs:simpleType name="telephone">
  <xs:restriction base="xs:string">
    <xs:length value="10" />
  </xs:restriction>
</xs:simpleType>
```

minLength et maxLength sont aussi possibles

```
<xs:simpleType name="celsiusWaterTemp">
 <xs:restriction base="xs:decimal">
  <xs:fractionDigits value="2"/>
  <xs:minExclusive value="0.00"/>
  <xs:maxExclusive value="100.00"/>
 </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name="civilite">
  <xs:restriction base="xs:string">
    <xs:enumeration value="M." />
    <xs:enumeration value="Mme" />
    <xs:enumeration value="Mlle" />
  </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name="codepostal">
  <xs:restriction base="xs:integer">
    <xs:totalDigits value="5" />
  </xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name="email">
  <xs:restriction base="xs:string">
    <xs:pattern value=".+@.+\..+" />
  </xs:restriction>
</xs:simpleType>
```

http://www.w3.org/TR/xmlschema11-1/#Simple_Type_Definitions

Déclaration d'éléments et attributs

- Un attribut est déclaré de la manière suivante :

```
<xs:attribute name="nomAttribut" type="xs:integer" use="required" default="0" />
```

  - Seul le nom est obligatoire
  - use : required ou optional (default pas possible)

- Un élément de type simple est déclaré via :

```
<xs:element name="nomElement" minOccurs="1" maxOccurs="1" type="xs:string" />
```

Définition de types complexes

- Un type complexe peut être composé
  - D'attributs
  - D'éléments
  - De texte
  - D'un mixte de tout cela (ou de rien...)
- La définition de type complexe est généralement associée à la déclaration d'un élément

Elément (simple) avec attribut

- Si on veut déclarer un élément avec attribut, on doit passer par un type complexe

```
<xs:element name="nomElement">
  <xs:complexType>
    <xs:attribute name="nomAttribut" type="xs:string" />
  </xs:complexType>
</xs:element>
```

```
<nomElement nomAttribut="aaa"/>
```

On peut aussi utiliser `xs:restriction`

```
<xs:element name="nomElement">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="nomAttribut" type="xs:positiveInteger" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

```
<nomElement nomAttribut="2">
bla
</nomElement>
```

Eléments composites

- On peut définir la composition d'un élément via
  - `<xs:sequence>` : les sous-éléments **doivent apparaître dans l'ordre**
  - `<xs:choice>` : seulement un des sous-éléments aux choix
  - `<xs:all>` : tous les éléments doivent apparaître dans un ordre quelconque

```
<xs:element name="personne">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="prenom" type="xs:string" minOccurs="1" maxOccurs="3"/>
      <xs:element name="nom" type="xs:string"/>
      <xs:choice>
        <xs:element name="portable" type="xs:string"/>
        <xs:element name="fixe" type="xs:string"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Eléments avec contenu mixte

- On veut mélanger texte et sous-éléments
  - Attribut mixed valué à true

```
<xs:element name="personne">
  <xs:complexType mixed=true>
    <xs:sequence>
      <xs:element name="prenom" type="xs:string"/>
      <xs:element name="nom" type="xs:string"/>
      <xs:choice>
        <xs:element name="portable" type="xs:string"/>
        <xs:element name="fixe" type="xs:string"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Autres fonctions XSD

- Les groupes

```
<xs:group name="groupname">
 <xs:sequence> <!-- all ou choice -->
    ...
 </xs:sequence>
</xs:group>
```

```
<!-- Dans la déf. d'un type -->
<xs:group ref="groupname"/>
```

```
<xs:attributeGroup name="groupname">
   ...
</xs:attributeGroup>
```

```
<!-- Dans la déf. d'un type -->
<xs:attributeGroup ref="groupname"/>
```

- Les clés et références

```
<xs:element name="bibliotheque">
  <xs:complexType>
  ...
  </xs:complexType>
  <xs:unique name="uniquelivre">  <!-- xs:key rend l'attribut obligatoire -->
    <xs:selector xpath="book" />
    <xs:field xpath="isbn" />
  </xs:unique>
</xs:element>
```

```
<xs:keyref name="refcleflivre" refer="cleflivre">
  <xs:selector xpath="citation"/>
  <xs:field xpath="ref" />
</xs:keyref>
```

Elément any et anyAttribute

- L'élément `<xs:any/>` est un jocker pour désigner n'importe quel élément

- L'attribut `<xs:anyAttribute/>` est un jocker pour désigner n'importe quel attribut

Comment associer un document XML et un schéma XSD ?

```xml
<?xml version="1.0"?>
<monDoc
xmlns="http://ic2a.upmf-grenoble.fr"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://imss-www.upmf-grenoble.fr/quelquepart.xsd">
  ...
</monDoc>
```

Exercice

Pour faire original :-)
Créez un schéma XML pour des types de documents XML représentant un CV (en fonction du CV créé précédemment).

Utiliser l'expressivité de XSD !!!

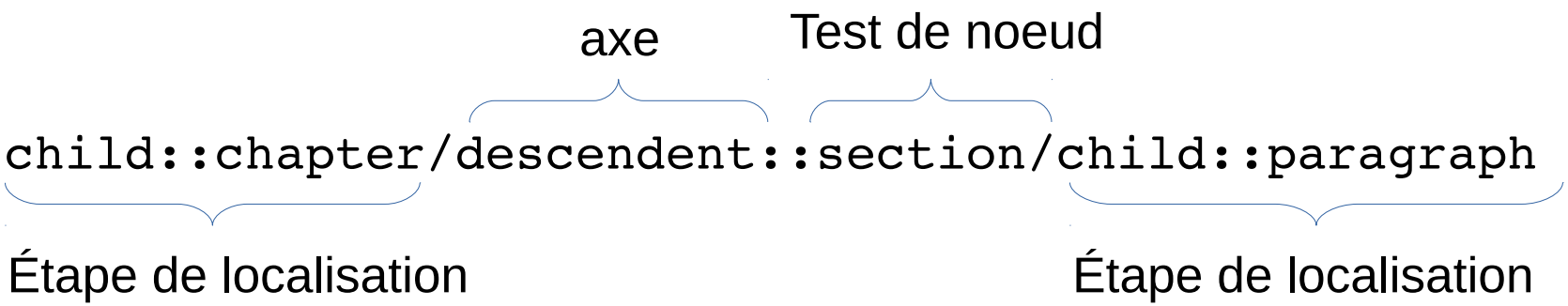
Validez le CV créé lors de l'exercice précédent.

XPath

- Pourquoi ?
  - Chercher, sélectionner, et extraire de l'information dans des documents XML
- XPath est une recommandation du W3C
- XPath n'est pas un langage XML
  - Mais un langage d'expression de chemins
- XPath travaille sur la représentation logique sous forme d'arbre des documents XML

Syntaxe XPath

- XPath permet d'exprimer des chemins de localisation :
  - Composé d'étapes de localisation
  - Séparées par des /



A finir...

XLink

- Xml est un langage pour le Web
- Le web est entre autre caractérisé par la possibilité de lier les pages entre elles par des hyperliens
- Comment ajouter des liens sur du XML ?
  - XLink : XML Linking Language
    - Recommandation du W3C : http://www.w3.org/TR/xlink/
    - Espace de noms : xmlns:xlink="http://www.w3.org/1999/xlink"

Liens XLink

- Tout élément XML peut être lié et devenir un lien
- Un lien peut être inter ou intra documents
- Il existent deux types de liens
  - Liens simples : liens unidirectionnels entre deux ressources (comme en HTML)
  - Liens étendus : plusieurs ressources peuvent être liées ensemble

Le liens simples

- 2 attributs principaux :
  - `xlink:href` : url de la cible du lien
  - `xlink:type` : type de lien (ici simple)

```
<?xml version="1.0" encoding="UTF-8"?>
<pagesweb xmlns:xlink="http://www.w3.org/1999/xlink">
  <page xlink:type="simple" xlink:href="http://www.upmf-grenoble.fr">Lien page UPMF</page>
  <page xlink:type="simple" xlink:href="http://www.w3c.org">Lien page W3C</page>
</pagesweb>
```

Autres attributs

- `xlink:show` : indique où ouvrir le lien quand il est activé
  - `new` (par défaut) : affiche le contenu de l'URL dans une nouvelle fenêtre
  - `replace` : affiche dans la fenêtre courante (remplace le contenu)
  - `embed` : inclus la ressource dans le document courant au niveau du lien
  - `other` : autre comportement défini dans une balise dédiée
  - `none` : ne rien faire
- `xlink:actuate` : indique quand activer le lien
  - `onLoad` : lien activé au chargement du contenu
  - `onRequest` (par défaut) : lien activé à la demande de l'utilisateur (click)
  - `other` : autre comportement défini dans une balise dédiée
  - `none` : ?
- `xlink:title` : court texte décrivant la ressource
- `xlink:role` : description/annotation/typage de la ressource distante

Remarque : un lien HTML équivaut à `actuate="onRequest" show="replace"`

Exemple

```
<?xml version="1.0"?>
<personne xmlns:xlink="http://www.w3.org/1999/xlink">
     <nom>Asimov</nom>
     <prenom>Isaac</prenom>
     <date-naissance>1920-01-02</date-naissance>
     <date-deces>1992-04-06</date-deces>
     <nationalite>Russe/Américain</nationalite>
     <webographie>
          <site-web lang="fr" xlink:title="ISAAC ASIMOV"
               xlink:role="Principal site sur Asimov en français"
               xlink:show="new"
               xlink:actuate="onRequest" xlink:type="simple"
               xlink:href="http://www.asimov.fr/" />
          <site-web lang="en" xlink:title="Isaac Asimov Home Page"
               xlink:role="Site majeur sur Asimov en anglais" xlink:show="embed"
               xlink:actuate="onLoad" xlink:type="simple"
               xlink:href="http://www.asimovonline.com/" />
     </webographie>
</personne>
```

Les liens étendus

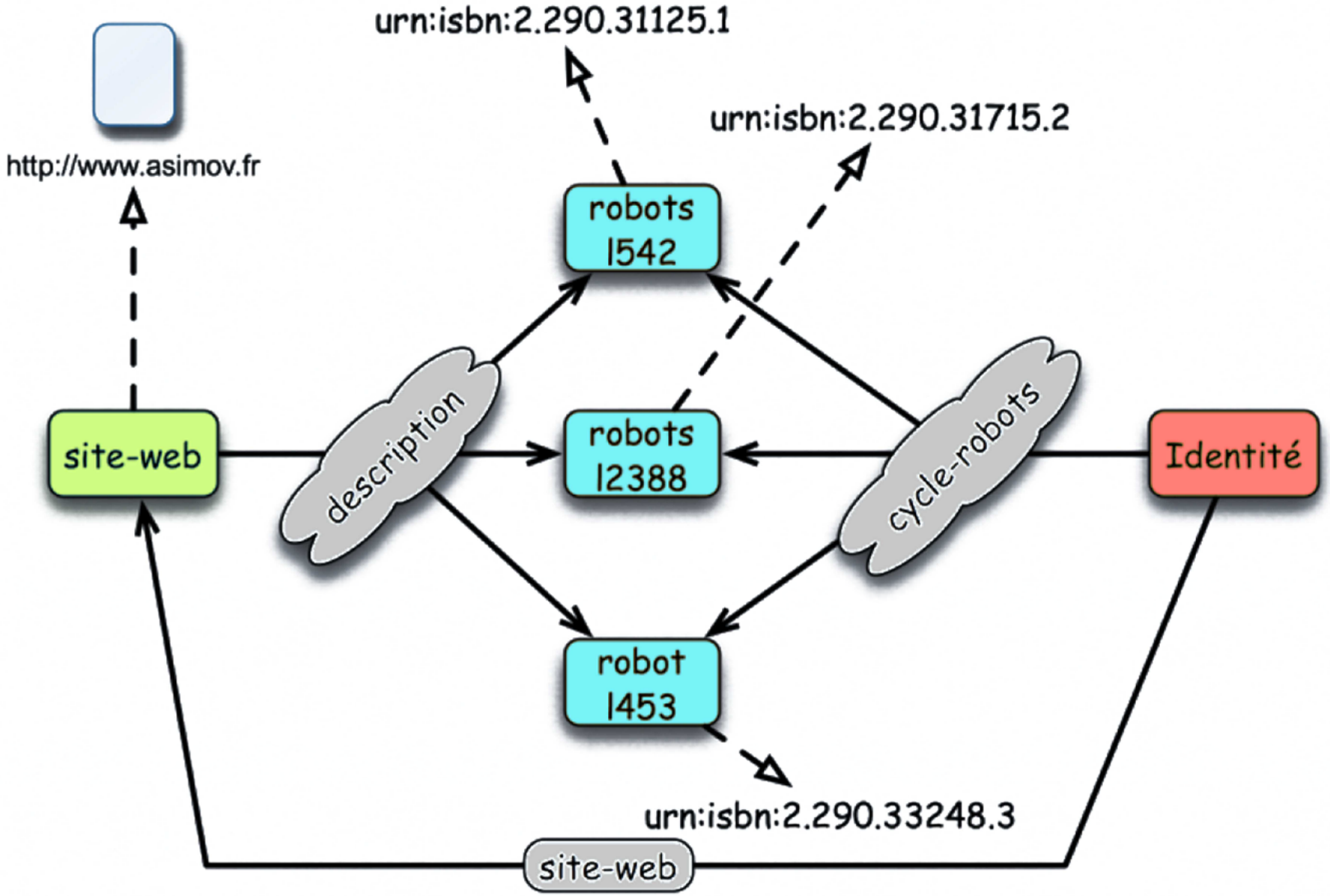
- Rappel
  - lien simple : lien unidirectionnel vers une seule ressource
  - lien étendu : associe un nombre arbitraire de ressources
- Un lien étendu est composé d'un élément XML (type=extended) englobant plusieurs autres éléments (type=locator|arc|title| resource) :
  - type=locator : la localisation des ressources distantes
  - type=arc : un mécanisme de traversée
  - type=title : une étiquette pour faciliter l'utilisation du lien par une personne
  - type=resource : les ressources locales associées par le lien

Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<auteur xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="extended">
    <identité xlink:type="ressource" xlink:label="identite">
        <nom>Asimov</nom>
        <date-naissance>1920-01-02</date-naissance>
    </identité>
    <site xlink:title="site d'un fan français" xlink:label="site-web"
        xlink:href="http://www.asimov.fr" xlink:type="locator" />
    <livre xlink:type="locator" xlink:href="urn:isbn:2.290.31125.1"
        xlink:label="robots" annee="1964" reference="l542">
        <titre>Un défilé de robots</titre>
    </livre>
    <livre xlink:type="locator" xlink:href="urn:isbn:2.290.34248.3"
        xlink:label="robots" annee="1950" reference="l453">

        <titre>I, robot</titre>
    </livre>
    <livre xlink:type="locator" xlink:href="urn:isbn:2.290.31715.2"
        annee="2002" reference="l2388">
        <titre>Le robot qui rêvait</titre>
    </livre>
    <cycle-robots xlink:type="arc" xlink:from="identite"  xlink:to="robots" xlink:title="début cycle robots" />
    <site-web xlink:type="arc" xlink:from="identite" xlink:to="site-web" />
    <description xlink:type="arc" xlink:from="site-web" xlink:to="robots" />
</auteur>
```

Exemple suite



XPointer

- Problème :
  - Comment référencer une partie d'un document XML dans des liens ?

    I.e faire un peut comme les ancres en HTML ?
- Solution : XPointer
  - Syntaxe :

    URI#xpointer(expression_xpath)

    URI#xpointer(expression_xpath1)xpointer(expression_xpath2)...
  - Exemple

    http://www.mon-site.fr/mon_exemple.xml#xpointer(//titre)

XInclude

- Objectif : mécanisme d'inclusion pour faciliter la modularité dans les documents XML
  - standard W3C : http://www.w3.org/2001/XInclude
  - différent de XLink + embed : XInclude intervient au niveau preprocessing
  - tous les parses ne gérent pas XInclude ; utilisation d'un « XInclude processor »
  - utilisation d'un espace de noms : `xmlns:xi="http://www.w3.org/2001/XInclude"`
- Syntaxe :

```
<xi:include href="document.xml" parse="xml" xpointer="xpointer(exp_xpath)"/>
```

  - `href` : URI définissant l'objet à inclure
  - `parse` : méthode d'inclusion pour savoir s'il faut ou non parser le document
    - `xml` → parser le document (valeur par défaut)
    - `text` → ne pas parser le document (pas d'interprétation des &, >, etc)
  - `xpointer` : si `parse="xml"`, alors XPointer identifie la partie du document XML à inclure
- Gestion d'erreurs : que faire en cas de problèmes (réseau HS, ressources inaccessibles, etc)
  - par défaut, le traitement de l'inclusion abandonne et génère une erreur
  - solution : définir un contenu à inclure quand <xi:include> génère une erreur
  - syntaxe : `<xi:fallback>...</xi :fallback>`

Xinclude
- exemple-

```xml
<?xml version="1.0" encoding="UTF-8"?>
<auteur>
   <identité>
      <nom>Asimov</nom>
      <prenom>Isaac</prenom>
      <date-naissance>1920-01-02</date-naissance>
      <nationalite>Russe/Américain</nationalite>
   </identité>

   <livre annee="1964" reference="1542">
      <titre>Un défilé de robots</titre>
   </livre>
   <livre annee="1950" reference="1453">
      <titre>I, robot</titre>
   </livre>
   <livre annee="2002" reference="12388">
      <titre>Le robot qui rêvait</titre>
   </livre>

</auteur>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<biblio xmlns:xi="http://www.w3.org/2001/XInclude">
   <xi:include href="ex_asimov.xml"
               parse="xml"
               xpointer="xpointer(//livre)">
      <xi:fallback>erreur</xi:fallback>
   </xi:include>
</biblio>
```

```
$xmllint --xinclude --noent --encode UTF-8 exemple.xml
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<biblio xmlns:xi="http://www.w3.org/2001/XInclude">

   <livre annee="1964" reference="1542">
      <titre>Un défilé de robots</titre>
   </livre>
   <livre annee="1950" reference="1453">
      <titre>I, robot</titre>
   </livre>
   <livre annee="2002" reference="12388">
      <titre>Le robot qui rêvait</titre>
   </livre>

</biblio>
```

Xinclude - exercice

- Ajoutez dans votre CV une instruction Xinclude qui permet d'ajouter au début de votre CV votre dernière expérience professionnelle...

XSL

- eXtensible Stylesheet Language
  - C'est une famille de recommandation du W3C
  - http://www.w3.org/Style/XSL/
- 3 parties
  - XSL Transformations (XSLT)
    - http://www.w3.org/TR/xslt
  - XML Path Language (XPath)
    - http://www.w3.org/TR/xpath/
  - XSL Formating Objects (XSL-FO)
    - http://www.w3.org/TR/xsl/

XSLT

- Langage de transformation de documents XML
  - permet de transformer des documents XML en d'autre documents XML
    - Par exemple : votre CV en XML vers une page XHTML
    - Mais peut également générer autre chose que du XML...
  - Basé sur XPath pour naviguer dans le document XML source
- Principe :
  - XPath permet de sélectionner un sous arbre sur lequel un template de transformation sera appliqué

Exemple

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="/persons">
    <root>
      <xsl:apply-templates select="person"/>
    </root>
  </xsl:template>

  <xsl:template match="person">
    <name username="{@username}">
      <xsl:value-of select="name" />
    </name>
  </xsl:template>
</xsl:stylesheet>
```

Feuille de style
style.xslt

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="style.xsl" ?>
<persons>
  <person username="JS1">
    <name>John</name>
    <family-name>Smith</family-name>
  </person>
  <person username="MI1">
    <name>Morka</name>
    <family-name>Ismincius</family-name>
  </person>
</persons>
```

Document XML

Feuille de style à appliquer

Résultat

```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <name username="JS1">John</name>
    <name username="MI1">Morka</name>
</root>
```

http://fr.wikipedia.org/wiki/Extensible_Stylesheet_Language_Transformations

XSLT – Syntaxe générale

- Racine :
  - `<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">`
  - Attributs version obligatoire + namespace
- Sortie :
  - `<xsl:output method="xml" indent="yes"/>`
  - Format de sortie
    - method : xml, html, text
    - indent : permet de mettre en forme le XML de sortie
    - encoding : encodage de la sortie
    - version : version du type de sortie
- Règles ou templates
  - `<xsl:template match="/persons">`
  - `<xsl:apply-templates select="person"/>`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl=
     "http://www.w3.org/1999/XSL/Transform"
     version="1.0">
  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="/persons">
    <root>
      <xsl:apply-templates select="person"/>
    </root>
  </xsl:template>

  <xsl:template match="person">
    <name username="{@username}">
      <xsl:value-of select="name" />
    </name>
  </xsl:template>
</xsl:stylesheet>
```

Les templates

- XSLT utilise des modèles de transformation
  - Appelés « template rules »
- Syntaxe d'une règle :

```
<xsl:template match="requete xpath sélectionnant un sous arbre dans le doc d'entrée">
    < !-- définition de la règle -->
</xsl:template>
```

- Fonctionnement :
  - Le processeur XSLT cherche le « meilleur » template parmi ceux qui satisfont l'attribut « match »
  - Ensuite le contenu du template est évalué

Evaluation des templates

- Si un élément n'est pas « matché »
  - i.e. aucune règle ne s'applique à ce nœud ni à aucun de ses parents
- Alors tout le texte de cet élément (et de ses descendants non matchés) est affiché

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persons>
  <person username="JS1">
    <name>John</name>
    <family-name>Smith</family-name>
  </person>
  <person username="MI1">
    <name>Morka</name>
    <family-name>Ismincius</family-name>
  </person>
  <dog>
      <name>dany</name>
  </dog>
</persons>
```

Essayez des documents XSLT :

- sans template
- un template qui ne matche que les éléments person

Sélection des templates

- Les éléments sont évalués à partir de la racine
- En cas de conflit de noms ?
  - i.e. deux templates sont possibles pour un nœud
  - On peut utiliser un attribut priority sur le template
  - Sinon la priorité par défaut dépend de la spécificité du patron xpath
    - Plus c'est spécifique, plus c'est prioritaire
    - Autrement c'est le dernier qui est appliqué
  - NB : il y a des cas plus compliqués
    - Templates importés par <xsl:include ...>

Priorité - exemples

- Exemples…
  - Démo en live

Les instructions de traitement

- Elles sont la pour :
  - Extraire des données du document d'entrée
    - `<xsl:value-of>`
  - Manipuler ces données
    - `<xsl:sort>`
  - Manipuler le parcours
    - `<for-each>, <xsl:if>, <xsl:choose>, <xsl:apply-templates>,`

L'élément value-of

- Il permet de recopier des données du document source dans la cible
  - `<xsl:value-of select="EXPRESSION XPATH" />`
  - Raccourci :

    `{@expression_xpath}`

```
<xsl:template match="person">
  <name username="{@username}">
    <xsl:value-of select="name" />
  </name>
</xsl:template>
```

L'élément for-each

- Permet de répéter un traitement sur un ensemble d'éléments
  -
```
<xsl:for-each select="EXPRESSION_XPATH" >
    <!-- Actions -->
</xsl:for-each>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/persons">
        <html>
        <body>
        <table>

        <xsl:for-each select="person" >
            <tr>
                <td><xsl:value-of select="family-name"/></td>
                <td><xsl:value-of select="name"/></td>
            </tr>
        </xsl:for-each>
        </table>
        </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

L'élément sort

- Permet de trier les éléments selon un critère donné
  -

```
<xsl:sort select="EXPRESSION_XPATH"/>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     <xsl:template match="/persons">
          <html>
          <body>
          <table>

          <xsl:for-each select="person" >
               <xsl:sort select="family-name"/>
                    <tr>
                         <td><xsl:value-of select="family-name"/></td>
                         <td><xsl:value-of select="name"/></td>
                    </tr>
          </xsl:for-each>
          </table>
          </body>
          </html>
     </xsl:template>
</xsl:stylesheet>
```

Les éléments « instructions de contrôle »

- L'élément <xsl:if> permet de tester une condition sur le nœud courant
  - ```
    <xsl:if test="xpath_condition"><!-- traitement --></xsl:if>
    ```
  - Exemple : http://www.w3schools.com/xsl/tryxslt.asp?xmlfile=cdcatalog&xsltfile=cdcatalog_if
- L'élément <xsl:choose> permet de spécifier des conditions multiple avec <xsl:when> et <xsl:otherwise>
  - ```
    <xsl:choose>
        <xsl:when test="xpath_condition><!-- traitement --></xsl:when>
        <xsl:otherwise><!-- traitement --></xsl:otherwise>
    </xsl:choose>
    ```
  - Exemple : http://www.w3schools.com/xsl/tryxslt.asp?xmlfile=cdcatalog&xsltfile=cdcatalog_choose

L'élément apply-templates

- Permet d'appliquer des templates à l'élément courant ou à ses descendants
  - Utilisé seul, il cherche à appliquer les templates sur tout le sous-arbre
  - Avec l'attribut select renseigné, cela permet de ne sélectionner un sous-ensemble
- Syntaxes :

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

```
<xsl:apply-templates select="xpath_expression"/>
```

Exemples

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="/persons">
    <root>
      <xsl:apply-templates select="person"/>
    </root>
  </xsl:template>

  <xsl:template match="person">
    <name username="{@username}">
      <xsl:value-of select="name" />
    </name>
  </xsl:template>
</xsl:stylesheet>
```

Testez !!!

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="/persons">
    <root>
      <xsl:apply-templates/>
    </root>
  </xsl:template>

  <xsl:template match="person">
    <name username="{@username}">
      <xsl:value-of select="name" />
    </name>
  </xsl:template>
</xsl:stylesheet>
```

Les variables

- XSLT est un langage « fonctionnel » donc sans effet de bord
  - Les variables sont à affectation unique
- Syntaxe :

```
<xsl:variable name="variable_name " select="xpath_expression">
    <!-- ou contenu de la variable -->
</xsl:variable>
```

- Deux moyens pour affecter
  - via xpath (select)
  - via dans le contenu
- Référence à une variable : `{$mavariable}`

Appel aux templates

- Les templates peuvent être appelés comme des fonctions
  - On donne un nom au template
    - `<xsl:template name='nom_du_template'> <!-- règle --> </xsl:template>`
  - Et on défini éventuellement des paramètres
    - `<xsl:param name='parameter_name' select='default_value'/>`
- Et puis on peut l'appeler :

```
<xsl:call-template name='nom_du_template'>
      <xsl:with-param name='parameter_name' select='parameter_value'/>
</xsl:call-template>
```

Copy-of

- Permet de copier le sous arbre à partir du nœud courant

```
<xsl:copy-of select="expression"/>
```

Autres fonctions et éléments

- Il y a plein d'autres fonctionnalités

  http://www.w3schools.com/xsl/xsl_w3celementref.asp
- Et des fonctions

  http://www.w3schools.com/xsl/xsl_functions.asp

Processeurs XSLT

- En ligne de commande – linux + unix
  - La libxslt : xsltproc
- Java
  - JAXP Transform API : inclus dans Java standard
    - https://docs.oracle.com/javase/tutorial/jaxp/xslt/index.html
- Dans les navigateurs web
  - Firefox : https://developer.mozilla.org/fr/docs/XSLT
    - Utilisation en JavaScript
  - Et dans les autres aussi

Exercices

- Pour s'entrainer :
  - https://class.stanford.edu/courses/DB/XSLT/SelfPaced/courseware/ch-querying_xml/seq-exercise-xml_countries_xslt_core/