Création de pages Web avec XML/XSLT

Luc Brun

Plan (1/3)

Plan (2/3)

Plan (3/3)

De quoi parle t'on

XML : eXtensible Markup Language
: Langage extensible de structuration de données.

XSL : eXtensible Stylesheet Language
: Langage de transformation.

Avantages :

- Standard libre du W3C (www.w3c.org)
- Prise en compte de nombreuses langues.
- Stockage de données au format texte $\rightarrow$ Flexibilité.

Stockage XML

Idéal pour :

- la manipulation de données (petites ou moyennes),
- l'archivage.

Utilisation croissante ayant atteint une certaine maturité.

- Traitements de textes : StarOffice & Open Office (zip de fichiers XML)
- Navigateurs : Netscape,
- Lecteurs de mail : Evolution,
- Éditeurs de diagrammes : Dia
- ...

Lien avec B.D.I.

données	XML	<cours> <intitule> BDI</intitule> <lieu> U11</lieu> <horaire> 14h-16h</horaire> </cours>
mise en forme	XSL	 Cours: BDI Salle : U11, 14h-16h
présentation	CSS	**Cours : BDI** *Salle : U11, 14h-16h*

Partie I

XML

eXtensible Markup Language

Un document XML

```
<?xml version="1.0"
 encoding="ISO-8859-1" ?>
<liste>
<ouvrage>
      <nom>
           10 sur 1 echelle de Richter
      </nom>
      <parution> 1999-01-01 </parution>
      <sujet> Science-Fiction </sujet>
      <auteur> Arthur C. Clarke</auteur>
      <auteur> Mike Quay(Mc)</auteur>
</ouvrage>
</liste>
```

Le préambule

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

- version : version de la norme XML utilisée par le document.
- encoding : type de caractères utilisés dans le document :

  UTF-8 : jeu de caractères universel

  ISO-8859-1 : (latin1) Europe occidentale, Amérique latine.

  ISO-8859- : (2) Europe centrale et orientale, (3) Europe du sud-est, (4) scandinavie, pays baltes, (5) Cyrillique, (6) Arabe,...

Les balises

Langage de balises : liste, ouvrage, sujet, auteur, nom : balises définies par l'utilisateur. Différentes combinaisons :

- `<balise> valeur</balise>`
  Exemple : `<auteur>Arthur C. Clarke</auteur>`
- `<balise> <sousbalise> valeur <sousbalise></balise>`
  Exemple : `<ouvrage> <auteur>Arthur C. Clarke</auteur></ouvrage>`
- `<balise attribut="valeur"></balise>`
  Exemple : `<auteur nationalite="française"></auteur>`

Combinaisons

```
<magazin>
   Micro Info
   <stock>
      <piece compaLinux="yes">
         USB DISK
      </piece>
      <piece compaLinux="no">
         Konika 200 Z
      </piece>
   </stock>
</magazin>
```

Remarque sur les attributs

```
<auteur nationalite="francaise">
Pierre bordage</auteur>
```

ou

```
<auteur>
Pierre bordage
<nationalite> francaise</nationalite>
</auteur>
```

Choix en fonction :

1. de la signification de l'attribut,
2. du contrôle que l'on souhaite exercer sur ses valeurs.

Caractères interdits

caractère	Entité
<	&lt ;
>	&gt ;
&	&amp ;
“	&quot ;
’	&apos ;

Remplacement des caractères non présents par leurs code (ex. € = &#8364)

liste des codes : selfhtml.selfhtml.com.fr/

Les DTD

XML ={ balises, relations entre balises, attributs, valeurs attributs}

Échange de documents XML ou maintenance à travers le temps de données stockées au format XML
$\Rightarrow$ nécessité de définir la structure du document XML et de la vérifier à chaque altération/création.

$\rightsquigarrow$ DTD

Exemple de DTD

```
<!ELEMENT liste (ouvrage)+ >
<!ELEMENT ouvrage (nom,parution,sujet,auteur+)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT parution (#PCDATA)>
<!ELEMENT sujet (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<liste><ouvrage>
        <nom> echelle de Richter </nom>
        <parution> 1999-01-01 </parution>
        <sujet> Science-Fiction </sujet>
        <auteur> Arthur C. Clarke</auteur>
        <auteur> Mike Quay(Mc)</auteur>
```

Éléments d'une DTD

(liste de sous balises)	
,	et : impose l'ordre des sous balises
|	ou : ordre quelconque
balise	une et une seule balise
balise ?	au plus une balise
balise+	au moins une balise
balise*	un nombre quelconque de balises
PCDATA	parsed character data : Une chaîne quelconque de caractères

DTD : Les attributs

```
<auteur nationalite="française">
  Pierre Bordage
</auteur>
<!ATTLIST auteur nationalite
(française|anglaise) #IMPLIED>
```

ATTLIST : Attributs de liste.

DTD : les attributs

```
<!ATTLIST balise attribut valeur option>
```

Valeur	Description
(valeur1|valeur2|...)	liste de valeurs autorisées pour l'attribut
CDATA	valeur quelconque
NMTOKEN	valeur quelconque, sans espace ni caractères spéciaux.

DTD : les attributs

```
<!ATTLIST balise attribut valeur option>
```

Option	Description
#REQUIRED	attribut obligatoirement présent
#IMPLIED	attribut éventuellement présent
"valeur"	valeur par défaut
#FIXED "valeur"	valeur constante

Lien XML/DTD

- DTD externes

```
<!DOCTYPE liste SYSTEM "ouvrage.dtd">
<liste>
    ⋮
</liste>
```

- DTD internes

```
<!DOCTYPE liste [
   la DTD
 ]>
<liste>
    ⋮
</liste>
```

Les espaces de noms

Fusion de plusieurs documents XML utilisant les mêmes balises $\Rightarrow$ Problème. Solution les espaces de noms (name spaces).
Exemple :

```
<perso:ouvrage>. . .</perso:ouvrage>
<emprunt:ouvrage>. . . </emprunt:ouvrage>
```

Déclaration d'un espace de nom

```
<biblio
xmlns:perso="http://www.univ-reims.fr/p.html"
xmlns:emprunt="http://www.univ-reims.fr/e.html"
>
```

- xmlns : XML name space.
- http ://www.univ-reims.fr/p.html : URI (Uniform Ressource Identifier) identificateur unique pour le nom. L'adresse n'a pas besoins d'exister.

Divers

- vérification de documents XML :
  xmllint fichier.xml - -dtdvalid fichier.dtd 1 > /dev/null
- Insérer du HTML dans un fichier XML :

```
<auteur
xmlns="http://www.w3.org/TR/REC-html40">
<html:p> <html:b>
    Pierre bordage
</html:b></html:p>
</auteur>
```

- Référence à un fichier CSS dans un fichier XML :

```
<?xml-stylesheet href="fichier.css"
type="text/css"?>
```

Définition d'un CSS lié à un fichier XML

Construire des règles CSS pour chaque balise ou des groupe de balises. Exemple :

```
liste {
font-family: sans-sherif;
font-size:12pt
}
auteur,sujet {
display:block
margin: 10px;
width: 400px;
}
```

Remarque : La référence à un CSS suffit pour que les balises HTML soient interprétées.

XSLT

eXtensible Stylesheet Language Transform

Un premier exemple

```
<?xml version="1.0"  encoding="ISO-8859-1" ?>
<xsl:stylesheet version='1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:output method="html" version="4.0"
encoding="ISO-8859-1" indent="yes"/>

<xsl:template match="/">
<html>
<head>
<title> Mon premier document XSLT </title>
</head>
<body>
<h1>Mon premier document XSLT </h1>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Remarques

1. Un document XSL est un document XML avec l'espace de nom xsl.
2. `xsl :output method='html'` format du fichier de sortie. Par défaut, method$\in${html,text,xml} (possibilité de produire du pdf avec xsl-foo)
3. `xsl :template match='/'` est une règle : élément fondamental en XSL. ('/' est l'élément racine appliqué au début du traitement d'un document XML)

Un exemple plus construit : XML

```
<?xml version="1.0"  encoding="ISO-8859-1" ?>
<!DOCTYPE liste SYSTEM "associations.dtd">
<associations>
<association departement="GIM">
<nom> ADEMI </nom>
<telephone> 03.26.21.81.81</telephone>
<membres>
<personne>
<nom>Jean Marie Leguerec</nom>
<fonction> Président</fonction>
</personne>
</membres>
</association>

...

</associations>
```

Un exemple plus construit : XSLT (1/2)

```
<?xml version="1.0"  encoding="ISO-8859-1" ?>
<xsl:stylesheet version='1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:output method="html" version="4.0"
encoding="ISO-8859-1" indent="yes"/>

<xsl:template match="/">
<html><head>
<title> Liste des associations</title>
</head><body>
<h1>Liste des associations</h1>
<table>
<xsl:apply-templates select="associations/association"/>
</table>
</body>
</html>
</xsl:template>
```

Un exemple plus construit : XSLT (2/2)

```
<xsl:template match="association">
        <tr>
                <td> <xsl:value-of select="nom"/>
                        <xsl:text>(</xsl:text>
                        <xsl:value-of select="@departement"/>
                        <xsl:text>)</xsl:text>
                </td>
                <td><xsl:value-of select="telephone"/> </td>
        </tr>
</xsl:template>
</xsl:stylesheet>
```

Résultat :

Liste des associations

ADMI (GIM)	03.26.21.81.81
⋮	⋮

Remarques

Principe de programmation XSLT :

Définir un ensemble de règles décrivant :

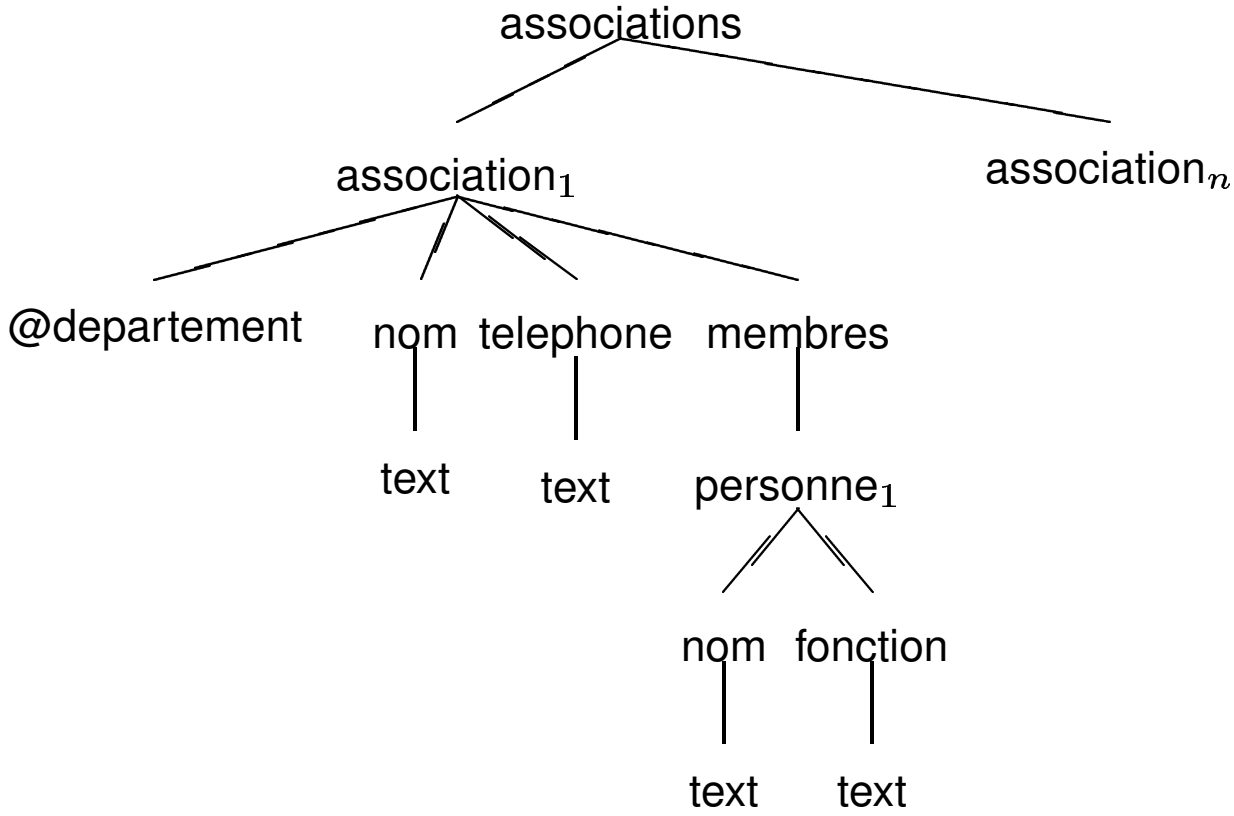
1. le traitement à appliquer sur les données,
2. l'agencement de ces traitements.

Comment définir les donnée sur lesquelles doivent s'appliquer une règle ?

```
<xsl :template match="toto">
```

toto : expression XPath

XPath : documents XML et arbres




```
<associations>
<association
departement="GIM">
<nom> ADEMI </nom>
<telephone>
03.26.21.81.81
</telephone>
<membres>
<personne>
<nom>
Jean Marie Leguerec
</nom>
<fonction>
Président
</fonction>
</personne>
</membres>
</association>
```

XPath : Chemins absolus

Syntaxe équivalente à celle des chemins de fichiers Unix.

1. Liste des noms d'associations
   `/associations/association/nom`
2. Liste des départements
   `/associations/association/@departement`
   Attention aux répétitions !
3. Liste des noeuds de type nom, petits fils de noeud association : `/associations/*/nom`
4. Liste des attributs des associations
   `/associations/association/@*`

XPath : Chemins relatifs

une règle XSLT évalue un noeud qui est le noeud courant de la règle.

1. Le noeud courant : .
   ex. `./text()` ⇔ `text()` noeud texte fils du noeud courant.
2. Le noeud père : ..
   ex. si membres noeud courant `../@departement` donne le département de l'association.
3. Liste des descendants réflexive : //
   ex. si association noeud courant `//fonction` tous les noeuds fonction descendant du noeud courant.

XPath : évaluation

L'arbre d'un document XML est parcouru par le processeur XSLT.

1. `following` : noeuds suivants dans l'évaluation,
2. `following-sibling` : noeuds suivants de même type (ex. association) dans l'évaluation.
3. `preceding` : noeuds précédents dans l'évaluation
4. `preceding-sibling` : noeuds précédents de même type dans l'évaluation.
5. fonction `position()` : position du noeud par rapport à l'ordre d'évaluation du noeud parent.
6. fonction last() : dernier noeud évalué pour le noeud parent.

XPath : Union de chemins

- Union :

```
//telephone | //nom
```

ensemble de tous les numéros de téléphones et de noms présents dans la base.

- Combinaison :

```
(//association | //personnes)/nom
```

ensemble des noms de personnes et d'associations.

XPath : conditions

Permet de sélectionner des noeuds en fonction d'un critère.
Exemples :

1. `associations/association[nom='ADEMI']` renvoi le (ou les noeuds) d'association dont le nom est ADEMI.
2. `/associations/association[@departement='GIM']/nom` renvoi les noms de toutes les associations du département GIM.

- Remarque : noeud[3]$\Leftrightarrow$ noeud[position()=3]
- Question : Nom des présidents de toutes les associations.

XPath : fonctions et opérateurs de base

opérateur	signification
&lt ;	<
&gt ;	>
=	égal
!=	différent
&lt ;=	≤
&gt ;=	≥
and,or	
not	non
number count(node-set)	nombre de noeuds

XPath : Fonctions sur les chaînes (1/2)

boolean contains(s_1,s_2)	s_1 contient s_2
contains(”01/01/1999”,“1999”)=true	
boolean start-with(s_1,s_2)	s_1 commence par s_2
start-with(”Mr dupond”,“Mr”)=true	
string concat(s_1,s_2,. . . ,s_n)	concatène s_1,..,s_n
concat(”né le”,“01/01/1999”)=”né le 01/01/1999”	
string substring-before(s_1,s_2)	chaine qui précède s_2 dans s_1
substring-before(“01/01/1999”,”/”)="01"	

XPath : Fonctions sur les chaînes (2/2)

string substring-after(s_1,s_2)	chaîne qui suit s_2 dans
substring-after("01/01/1999","/")="01/1999"	
string substring(s_1,n,l ?)	chaîne de longueur commençant à la po tion n
substring("01/01/1999",7)="1999"	
number string-length(s)	longueur de la chaîne.
string-length("01/01/1999")=10	
string normalize-space(string ?)	normalise les espac (avant,milieu, début)
normalize-space(" il était ")="il était"	

XPath : fonctions numériques

sum(node-set)	somme d'un nombre de noeuds
floor	floor(4.4)=4
ceil	ceil(4.4)=5
round	round(4.4)=4 round(4.6)=5

Exercices (1/2)

- Sélections des associations des départements GIM et INFO,
- Sélection de toutes les associations sauf celles du département GIM,
- Sélection de toutes les associations comportant au moins 5 membres,
- Sélection de toutes les associations du département GIM dont Mr Leguerec n'est pas membre,
- Sélection de toutes les associations du département GIM dont le nom du président contient Leguerec.

Exercices (2/2)

Écrire un programme XSLT qui affiche la liste des associations sous la forme :

Liste des associations

1. ADEMI, Département GMI
   Liste des membres
   - Jean Marie Leguerec (Président)

   ⋮

XSLT : importation de documents

- Importation de documents XSLT
  - `<xsl :import href="url"/>` : les règles du fichiers importé ont une priorité inférieure à celle du fichier qui inclut $\Rightarrow$ possibilité de redéfinir un programme.
  - `xsl :include href="url"/>` les règles du fichier inclu et courant ont la même priorité.
- Importation de documents XML : fonction document exemple :

```
<xsl :value-of
select="document('config.xml')/langs[@default='true']
```

charge la langue par défaut dans config.xml
$\Rightarrow$Possibilité de travailler sur plusieurs documents XML.

XSLT : Le tri

Utilisé uniquement pour `<xsl :apply-template>` et `<xsl :for-each>` (voir plus loin).

```
<xsl:sort select="expr"
lang="nmtoken"
data-type="text|number"
order="ascending|descending"
case-order="upper-first|lower-first"/>
```

select : critère de tri

lang : langue,

data-type : texte ou nombres,

order : ordre du tri,

case-order priorité majuscules/minuscules.

XSLT : exemple de tri

Tri des associations par département puis par nom.

```
<xsl:apply-template select="association">
<xsl:sort select="@departement"/>
<xsl:sort select="nom"/>
</xsl:apply-template>
```

Possibilité de tri multiples.

XSLT : les paramètres (définition)

```
<xsl:param name="nom"
select="default value"/>
default value
</xsl:param>
```

- Au début d'un document correspond à un paramètre du programme (équivalent de argv en C).
- Dans la définition d'une règle correspond à un paramètre de celle-ci.
- Accès à la valeur d'un paramètre par
  `<xsl :value-of select="$nom" />`

XSLT : les paramètres (appel)

```
<xsl:with-param name="nom"
select="default value"/>
default value
</xsl:with-param>
```

Exemple :

```
<xsl:apply-template select="association">
<xsl:with-param name="lang" select="fr"/>
</xsl:apply-template>
```

ou

```
<xsl:apply-template select="association">
<xsl:with-param name="lang">
<xsl:value-of
    select="document('config.xml')/langs[@default='true']
</xsl:with-param>
</xsl:apply-template>
```

XSLT : Les règles (définition)

```
<xsl:template match="expr xpath"
name="nom"
priority="number"
mode="nom">
</xsl:template>
```

name : défini une règle nommée $\approx$ procédure.

priority : par défaut la règle la plus prioritaire est la dernière définie $\Rightarrow$ Possibilité de forcer des priorités.

mode : permet de regrouper des règles en familles. ex. mode="item" (représentation des associations sous forme d'items) mode="table" (représentation sous forme de tables).

XSLT : les règles (appels 1/2)

Les règles non nommées :

```
<xsl:apply-template select="expr xpath"
mode="nom">
<xsl:sort> ou
<xsl:with-param>
</xsl:apply-template>
```

- `<xsl :apply-template/>` applique les règles sur tous les fils du noeud courant : dangereux
- `<xsl :apply-template select="document('config.xml')/langs"/>` applique une règle à partir d'un fichier XML externe.

XSLT : les règles (appels 2/2)

Les règles nommées :

```
<xsl:call-template name="nom"
<xsl:with-param> (0 à n)
</xsl:call-template>
```

Exemple :

```
<xsl:call-template name="display_title">
<xsl:with-param name="title"
select="Premier appel de règle nommée"/>
</xsl:call-template>
```

XSLT : Les boucles

Boucle sur un ensemble de noeuds.

```
<xsl:for-each select="expr xpath">
<xsl:sort> (0 à n) contenu
</xsl:for-each>
```

Exemple :

```
<ul><xsl:for-each select="membres">
<xsl:sort select="nom">
<li>
<xsl:value-of select="nom"/>
<xsl:text>(</xsl:text>
<xsl:value-of select="fonction"/>
<xsl:text>)</xsl:text>
</li>
</xsl:for-each></ul>
```

XSLT : expressions conditionnelles (1/2)

L'instruction if :

```
<xsl:if test="expr xpath">
contenu
</xsl:if>
```

Exemple :

```
<xsl:if test="not(@departement)">
        <xsl:text>
                Département non spécifié
        </xsl:text>
</xsl:if>
```

Pas de else...

XSLT : expressions conditionnelles (2/2)

L'instruction choose :

```
<xsl:choose>
 <xsl:when test="expr xpath">
 </xsl:when> (1 à n)
 <xsl:otherwise>  </xsl:otherwise>(0 à 1
</xsl:choose>
```

Exemple :

```
<xsl:choose>
 <xsl:when test="@departement">
     <xsl:value-of select="@departement"/>
 </xsl:when>
 <xsl:otherwise>
         <xsl:text> Département non spécifié</xsl:text>
 </xsl:otherwise>
</xsl:choose>
```

XSLT : insertion d'éléments

Cas simple : insertion directe `<ul>...</ul>`.
Utilisation d'attributs ou de noms variables

⇒`<xsl :element>`

```
<xsl:element name="nom"
use-attribute-sets="names">
<xsl:attribute name="nom">
contenu
</xsl:attribute> (0 à n)
contenu
</xsl:element>
```

XSLT : exemple d'insertion d'éléments

```
<link name="mon code source"> (name non obligatoire)
http://www.univ-reims.fr
</link>
```

But : `<a href=''http ://www.univ-reims.fr> mon code source </a>`

```
<xsl:template match="link">
  <xsl:element name="a">
    <xsl:attribute name="href">
      <xsl:value-of select="."/>
    </xsl:attribute>
    <xsl:choose>
      <xsl:when test="@name">
        <xsl:value-of select="@name"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text> url </xsl:text>
      </xsl:otherwise>
  </xsl:element>
</xsl:template>
```

XSLTPROC

```
xsltproc [options] file.xsl file.xml > file.htm
```

Options :

- **–output ou -o file** nom du fichier de sauvegarde
- **–param** nom noeud
- **–stringparam** nom valeur