XML et XML schéma : représentation des données

Gérard Florin
Laboratoire CEDRIC
CNAM Paris

La représentation des données

Origines : les formats de documents SGML, HTML

SGML (‘Standard Generalized Markup Language’) - 1986

- Un langage de balisage **très général** pour définir des documents électroniques, **indépendants de la forme visualisée**.
- Définir le contenu d'un document (rapport technique, livre, courrier, …).
- Un contenu est **transformé en autant de formats imprimés** que nécessaire**.**
- Définir par ailleurs la forme imprimée.
- Le but principal reste de **transmettre, stocker des documents à imprim**er avec économies de volume et souplesse de formatage.
- **SGML est complexe => son usage est restreint à des spécialistes de la gestion documentaire.**

HTML (‘HyperText Markup Language’) - 1989

- **Développement de la documentation accessible en ligne**

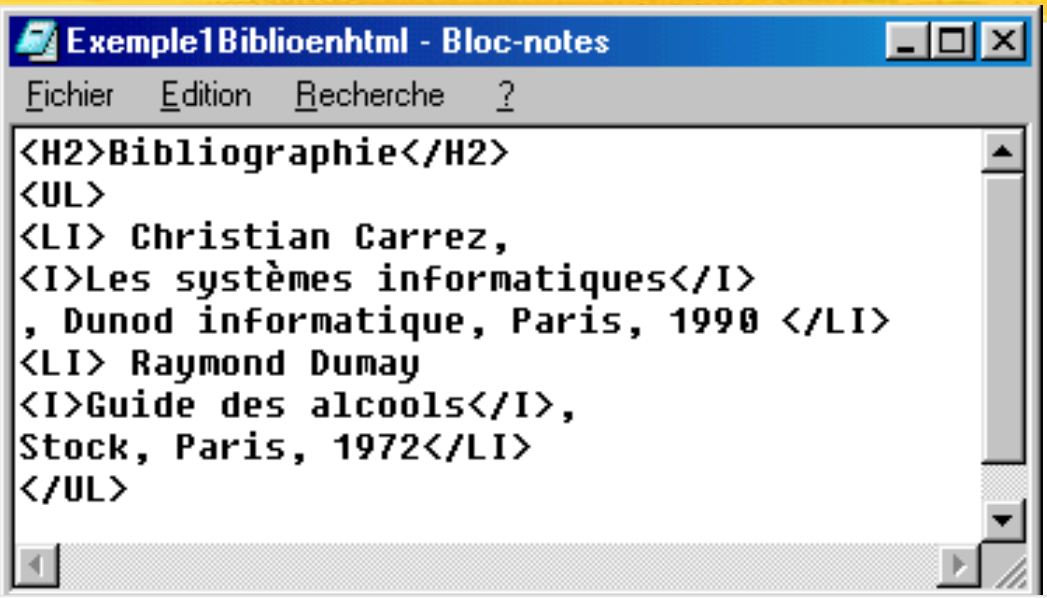
  => Avantage des liens **hypertextuels**.
- Première études sur **le langage de balises hypertextes HTML** 1989 conçu à partir de SGML comme une version très simplifiée.
- Disponibilité et **première normalisation** en 1992
- **Démonstration de la très grande capacité** d'accès à l'information rendue possible par le WEB (HTML + HTTP + URL).

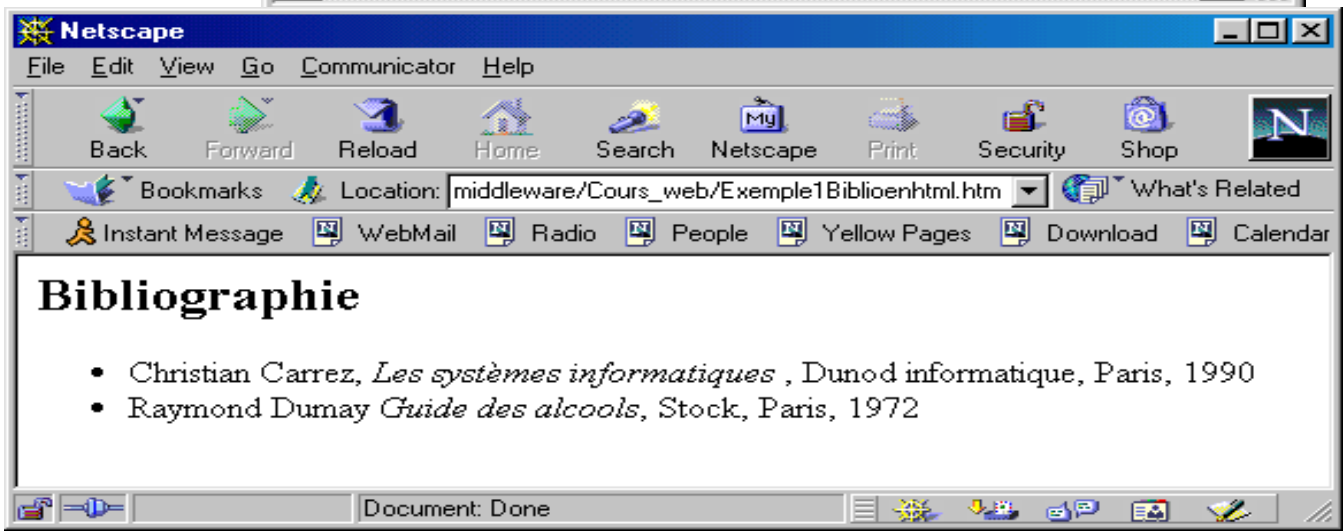
Quelques caractéristiques de HTML

- HTML est un **langage de balises** comme SGML :
  - Exemple <H2>
- Normalisation **d'un ensemble figé** de symboles encadrés par "<" et ">".
- Les balises HTML définissent **des directives de mise en page du text**e encadré par un couple balise ouvrante, balise fermante:
  - **<H2> xxxx </H2>** un titre de niveau 2,
  - **<UL> yyyy </UL>** une liste d'items (sans numéros d'item),
  - **<LI> zzzzzz </LI>** un item de la liste,
  - **<I> ttttttttt </I>** un texte en italique.
- **Pas de type de document** (pas de structuration) prédéfinie permettant de définir abstraitement une structure de document

Exemple d'une bibliographie en HTML:

- Une structuration de document complètement **orientée vers la présentation graphique.**





Discussion des choix de HTML (1)

- **HTML: un langage trop graphique**
  - A l'origine HTML définissait des balises **indépendantes de la représentation** graphique des informations (Ex: UL définit une liste).
  - Logiquement, la présentation graphique d'une liste ne doit pas être toujours la même: elle **dépend d'une adaptation du navigateur à un contexte**.
    - Ex: présenter des informations en Braille pour des non voyants.
  - Au fil du temps les fabricants ont introduits pour leurs **besoins des éléments graphiques**. (Ex: I pour italique, FONT, CENTER, BGCOLOR. De même UL a pris le sens d'une indentation, pas d'une liste.
- **Le balisage HTML est devenu spécifique des différents navigateurs pour les besoins d'affichage**

  => HTML est en fait un langage graphique pour Explorer ou Mozilla Firefox.

Discussion des choix de HTML (2)

- **HTML: un langage non extensible**
  - Le langage HTML a été conçu pour être **assez simple** de sorte que le nombre et la signification des balises est **limité**.
    - Ex: Il n'existe pas de balisage pour la représentation des données en chimie (molécules, formules, valeurs numériques)
  - Le langage HTML est **figé**.
    - Toutes les balises utilisables sont définies au départ ce qui est intenable si l'on voulait prendre globalement en compte les besoins d'un grand nombre de métiers.
- **HTML est figé et assez simple : pas de possibilité pour un utilisateur de base d'introduire de nouvelles balises.**

Discussion des choix de HTML (3)

- **HTML: un langage de description de documents non structurés**
  - HTML permet de définir de façon beaucoup trop **limitée** la **structure** d'un document.
  - Il n'y a en fait pas de **vérification d'une structure** pour le document que l'on peut définir.
    - Ex: on peut créer un document commençant par une tête de chapitre H2 et poursuivant par une tête de chapitre H1.
- **HTML définit en fait un univers de documents plats.**
  - Une recherche doit considérer un document HTML comme **une chaîne de caractères**.
  - Pas de moyen de partager entre communicants **une structure de document** préétablie.
- **HTML ne type pas les documents.**

Conclusion : avenir de HTML

- **HTML peut rester très longtemps utilisé comme langage graphique pour les navigateurs WEB.**
- **HTML présente des limitations trop importantes pour rester à terme le support de la représentation des données échangées sur le WEB.**
  - Non **séparation** du document et de sa présentation graphique.
  - Non **extensibilité**.
  - Non **structuration**, **typage**.

XML et le typage par les DTD

Introduction XML : Héritage SGML/HTML

- **SGML** riche, lourd, mal adapté au Web.
- **HTML** adapté à la présentation graphique, mais limité par un ensemble de balises figé, non extensible et sans typage.
- **Groupe de travail XML** à partir de août 1996 qui est composé essentiellement de membres du groupe de travail SGML.
  - **Recherche d'un langage assez simple** mais présentant une richesse proche de SGML.
  - **XML : un sous-ensemble de SGML**, qui élimine des points trop ciblés sur certains besoins.
  - Un document XML est **conforme** SGML.

Conséquence sur XML

- **Extensibilité**: pouvoir définir de nouvelles balises.
- **Structuration** : pouvoir modéliser des données d'une complexité quelconque.
- **Validation** : pouvoir vérifier la conformité d'une donnée avec un type (un modèle de structure).
- **Indépendance du média**: pouvoir formater un contenu selon des représentations diverses.
- **Interopérabilité** : Pouvoir échanger et traiter une donnée en utilisant de nombreux types de logiciels.

Caractéristiques de XML (1) : un langage de niveau méta

- **XML n'est pas un langage de balises de plus** (comme HTML).
- **XML permet de définir de nouveaux langages de balises** (comme SGML).
  - **Notion de langage de niveau méta**
- En quelque sorte, un langage qui permet de **définir des langages**.

Caractéristiques de XML (2) : un langage verbeux

Choix délibéré: XML définit tout en format caractère (lisible par un être humain).

- **Avantage :** pour le **développement** des codes, **la mise au point, l'entretien** et pour **l'interopérabilité** (choix des caractères UCS 'Universal Character Set' ISO Unicode).
- **Inconvénient:** Les données codées en XML sont **toujours plus encombrantes** que les mêmes données codées en binaires.
  - L'espace mémoire, l'espace disque et la bande passante sont devenus **moins chers**.
  - Les **techniques de compression** sont accessibles **facilement et gratuitement**.
  - Elles sont applicables **automatiquement** dans les communications par modems ou en HTTP/1.1 et **fonctionnent bien** avec XML.

Structure d'un document XML (1) : structure logique

Un document XML est composé de trois parties:

- **Un prologue** qui comporte:
  - Des déclarations diverses
  - Des instructions de traitement (optionnelles)
  - Une déclaration de type du document (optionnelle)
- **Un ensemble d'éléments** organisés en arbre (les balises).
- **Des commentaires** <!-- Un commentaire -->

Structure d'un document XML (2) : structure logique - prologue

- Exemple de déclaration XML :
  - <?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
    - version : version du XML utilisée dans le document
    - encoding : jeu de codage de caractères utilisé
      - Jeu de caractères habituel pour le français : *ISO-8859-1, a* tendance à être remplacé par l'ISO-8859-15
      - Par défaut : UTF-8.
    - standalone : dépendance du document par rapport à une dtd
      - Si standalone = no, le processeur de l'application attend une dtd
      - Valeur par défaut = no.
- La déclaration est facultative, mais préférable

Structure d'un document XML (3) : structure logique - prologue

- Une instruction de traitement est une instruction interprétée par l'application servant à traiter le document XML

  <?xml-stylesheet type="text/xsl" href="bib.xsl"?>

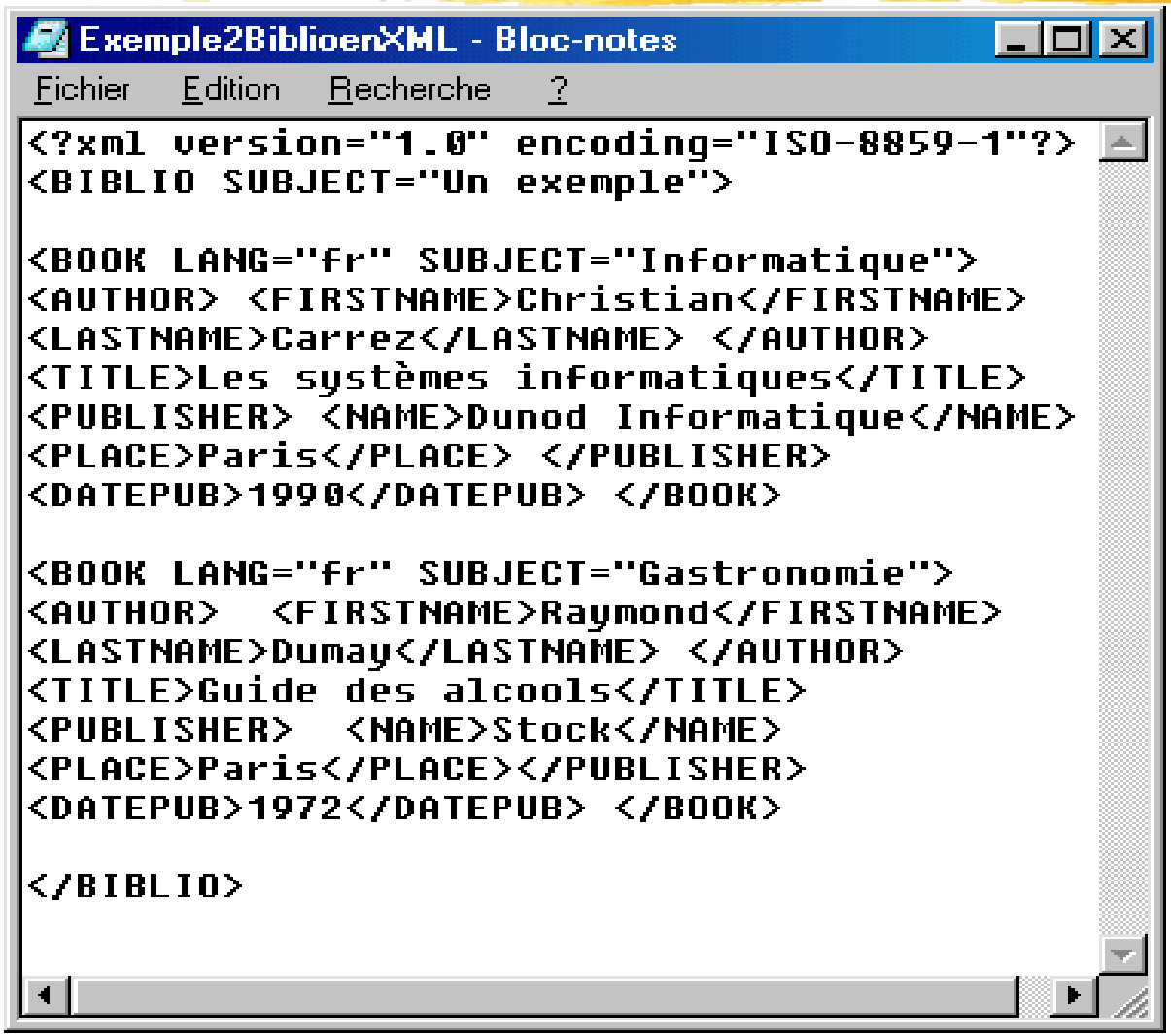
  - Application : xml-stylesheet
  - Type : type de fichier
  - href : URL du fichier

Corps d'un document XML

- **XML définit un cadre pour baliser n'importe quel document structuré en arbre (balises ou 'tags')**
- **Notion d'élément XML (associé à une balise)**: définit une donnée sous la forme d'une chaîne de caractères comme ayant un sens (message, de, pour, objet, texte)
- **Notion d'attribut XML (d'un élément)**: pour associer une donnée à une balise (localisation, codage).
- **Exemple simple:**

```
<message securite="secret défense">
        <de>Jean</de>
        <pour>Jacques</pour>
        <objet>Rappel</objet>
        <texte>On se voit demain à 10h</texte>
</message>
```

Exemple d'une bibliographie en XML



Arbre associé à une bibliographie (1)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <BIBLIO SUBJECT="Un exemple">
  - <BOOK LANG="fr" SUBJECT="Informatique">
    - <AUTHOR>
        <FIRSTNAME>Christian</FIRSTNAME>
        <LASTNAME>Carrez</LASTNAME>
      </AUTHOR>
      <TITLE>Les systèmes informatiques</TITLE>
    - <PUBLISHER>
        <NAME>Dunod Informatique</NAME>
        <PLACE>Paris</PLACE>
      </PUBLISHER>
      <DATEPUB>1990</DATEPUB>
    </BOOK>
  - <BOOK LANG="fr" SUBJECT="Gastronomie">
    - <AUTHOR>
        <FIRSTNAME>Raymond</FIRSTNAME>
        <LASTNAME>Dumay</LASTNAME>
      </AUTHOR>
      <TITLE>Guide des alcools</TITLE>
    - <PUBLISHER>
        <NAME>Stock</NAME>
        <PLACE>Paris</PLACE>
      </PUBLISHER>
      <DATEPUB>1972</DATEPUB>
    </BOOK>
  </BIBLIO>
```

Structure d'un document XML (2) : structure physique

- Document = arbre :
  - une racine **unique**
    - Englobe *tous* les autres éléments
    - Est ouvert après le prologue
  - Eléments
    - Branches ou feuilles
    - **Contiennent du texte ou d'autres éléments enfants**
    - ou sont vides : ils ne contiennent pas d'élément-enfant.
  - **Attributs**
    - Tous éléments peut contenir un ou plusieurs attributs dans la balise *ouvrante*
    - Chaque élément ne peut contenir qu'une fois le même attribut.
    - Un attribut est composé d'un nom et d'une valeur.

Structure d'un document XML (2) : structure physique

- Une entité peut référencer une autre entité pour forcer **l'inclusion** à la place dans le document.
- Une entité peut être **nommée** (avoir un alias).
- Une entité peut être **interne** (son nom est alors un raccourci pour sa valeur).
- Une entité peut être **externe** et représenter un fichier local ou distant contenant du texte XML.

Structuration d’un document

- Cet exemple est tiré de http://www.gchagnon.fr/cours/xml/exercices/exo1.html
- Une bouteille d'eau Cristaline contient par litre 71 mg d'ions positifs calcium, et 5,5 mg d'ions positifs magnésium. On y trouve également des ions négatifs comme des chlorures à 20 mg par litre et des nitrates avec 1 mg par litre. Elle est recueillie à St-Cyr la Source dans le Loiret. Son code barre est 3274080005003.

Structuration d'un document

- <bouteille>
  - <marque>Cristaline</marque>
    - <composition>
      - <ion type="positif">calcium 71mg/l</ion>
      - <ion type="positif">magnésium 5,5mg/l</ion>
      - <ion type="negatif">chlorure 20mg/l</ion>
      - <ion type="negatif">nitrate 1mg/l</ion>
      - <autre type="metal">fer</autre>
- </composition>
- <source>
  - <ville>St-Cyr la Source</ville>
  - <departement>Loiret</departement>
- </source>
- <code_barre>3274080005003</code_barre>
- </bouteille>

Documents bien formés

- **Si un document XML respecte les règles de la grammaire XML on dit qu'il est bien formé.**
- **Les règles d'un document bien formé**
  - Toute balise **ouverte** doit être **fermée**, Ex: <livre> </livre>
  - L'ensemble des balises est **correctement imbriqué**. Ex : Entrelacement mal formé   ….
  - Les valeurs d'attributs sont entre **guillemets**"
- Les **balises uniques correspondent à des documents vides** et sont notées: Ex : <IMG source="image.gif"/>
- Les caractères < & sont notés < &
- Un document commence par une **déclaration XML**
  - <?xml version="1.0" encoding ="iso-8859-1" standalone="yes"?>

Documents valides : DTD

- **Déclaration de type de document : DTD 'Document Type Definition** ’ une notion héritée de SGML permettant la définition formelle d’un type de document.
- Une information de niveau méta pour qu'un analyseur puisse vérifier la **conformité** d’un document à une spécification de type (notion de document **valide**).
  - **intitulé** des balises et de leurs **attributs**,
  - **imbrication** des balises (structure d’arbre du document),
  - **caractère obligatoire/facultatif** de certaines infos**.**
- Pour un document XML une DTD est **optionnelle**:
  - **En son absence**: on constate une structure.
  - **En sa présence**: on contraint la conformité d'un document XML.

Stockage de la DTD (1)

- 2 endroits de stockage (interne/externe) :
  - Stockage interne: déclaration écrite à l'intérieur du prologue du document XML

```
<?xml version="1.0" standalone="yes"?>
    <!DOCTYPE biblio[
     <!ELEMENT biblio (livre)*>
     <!ELEMENT livre (titre, auteur, nb_pages)>
       <!ATTLIST livre type (roman | nouvelles | poemes |
     théâtre) #IMPLIED
         lang CDATA "fr"
       >
     <!ELEMENT titre (#PCDATA)>
     <!ELEMENT auteur (#PCDATA)>
     <!ELEMENT nb_pages (#PCDATA)>
     ]>
    ...
```

Stockage de la DTD (2)

- Stockage externe : la DTD est stockée dans un fichier à part et peut donc être partagée entre plusieurs documents XML
  - Exemple d'une DTD privée :
    - <?xml version="1.0" encoding="ISO-8859-1"?>
      <!DOCTYPE biblio **SYSTEM** "**...**" "biblio.dtd">
      <biblio> **...** </biblio>…
  - Exemple d'une DTD publique :
    - <!DOCTYPE html **PUBLIC** "**-**//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

DTD : Déclarations de types d'éléments de base (1)

- <!**ELEMENT** nom_élément **(**contenu_élément**)** >
  - **Le nom de l'élément** le nom de la balise dans un document.
  - Contenu: **un type de base** (chaîne de caractères avec des variantes) ou un type construit (en fait une séquence d'éléments).
    - **EMPTY : Chaîne vide**
      - <! ELEMENT elem (EMPTY)>
      - Eléments vides: support d'attributs
    - **#CDATA : Chaîne non analysée**
      - La chaîne est supposée contenir des données qui n'ont pas à être parsées.
    - **#PCDATA : ('Parsed Character DATA')**
      - Chaîne de caractères devant être parsée.
    - **ANY : Une chaîne quelconque.**

DTD : Déclarations de types d'éléments construits (2)

- **Élément avec fils : Types d'éléments construits à partir d'autres types d'éléments (types article).**
- On peut préciser la structure (séquence) des fils au moyen **de constructeurs.**
  - Nom_élément — une seule occurrence
  - Nom_élément***** — 0, 1 ou n occurrences
  - Nom_élément**?** — 0 ou 1 occurrence (valeur optionnelle)
  - Nom_élément**+** — 1 ou n occurrences
  - Nom1**|**Nom2 — alternative
  - **,** **( )** — séquence et parenthèses
- **Types éléments 'mixtes' :** des éléments et des chaînes

Exemples – indicateurs d'occurrence (1)

- Exemple de séquence : un mémoire

  <!ELEMENT mémoire (titre, résumé, remerciements?, introduction, chapitre+, conclusion, references) >

  - Un mémoire est constitué d'un titre (un seul titre, sa présence est obligatoire), d'un résumé (même chose que pour le titre), d'au plus un chapitre (soit on a un chapitre, soit on n'a pas de chapitre), d'une conclusion et de reference
  - Un mémoire est défini comme une séquence ordonnée. Il **doit** être ordonné de la façon suivante : titre, résumé, remerciements, introduction, chapitre, conclusion, references
- Exemple de choix avec indicateurs d'occurence par élément

  <!ELEMENT elt0 (elt1* | elt2* | elt3*)>

  - Instanciations valides :
    - <elt0>
      <elt2>(...)</elt2>
      <elt2>(...)</elt2>
      </elt0>
  - Instanciations non valides :
    - <elt0>
      <elt3>(...)</elt3>
      <elt2>(...)</elt2>
      </elt0>
    - <elt0>
      <elt2>(...)</elt2>
      <elt3>(...)</elt3>
      </elt0>

Exemples – indicateurs d'occurrence (2)

- <!ELEMENT elt0 (elt1 | elt2 | elt3)*>
  - Choix d'éléments avec indicateur d'occurence global :
  - Exemple valide :
    - <elt0>
      <elt2>(...)</elt2>
      <elt3>(...)</elt3>
      <elt1>(...)</elt1>
      </elt0>
- <!ELEMENT citation (#PCDATA | auteur)*>
  - Pas de contrainte sur l'ordre d'apparition des éléments
  - Déclaration la plus souple possible
  - Exemple :
    - <citation>
      <auteur>Shakespeare</auteur>Être ou ne pas être
      </citation>

DTD : Déclarations de types d'attributs

- **Les listes d'attributs 'attlist'** définissent pour un élément la liste des attributs de cet élément avec leurs propriétés.
  - Nom de l'attribut, Type de l'attribut, Valeur par défaut.
    - <!ATTLIST nom_element
    - nom_attribut1 type1 défaut1
    - …
    - nom_attributn typen défautn>
- **Le paramètre défaut**
    - **#REQUIRED**:attribut obligatoire à fournir.
    - **#IMPLIED** : attribut optionnel.
    - **#FIXED 'value'**: attribut de valeur invariable égale à 'value'.
    - **'value'**: une valeur pour l'attribut quand elle n'est pas donnée.

Exemple

- <!ELEMENT elt (...)>

  ...

  <!ATTLIST elt attr CDATA #IMPLIED>
- <elt attr=« blabla">(...)</elt>

DTD : Typage des d'attributs (1)

- Trois catégories de types: chaînes, listes de chaînes, valeurs énumérées.
- **CDATA :** Les valeurs possibles sont des chaînes de caractères non analysées par le parseur.
  - Ex: <!ATTLIST fichier chemin CDATA #REQUIRED>
- **ID** : Les valeurs sont des identifiants uniques.
  - Ex: <!ATTLIST dossier ident ID #REQUIRED nom CDATA #IMPLIED>

Conclusion XML

- **Multiples avantages**
  - **Un langage effectivement simple**.
  - XML rend possible l'arrivée d'une nouvelle génération **d'outils logiciels** pour des **plate-formes hétérogènes**.
    - de manipulation,
    - de transmission,
    - de visualisation de données distribuées.
- **Les inconvénients**
  - La simplicité de base conduit à **énormément de compléments de toutes natures** => apparition de dialectes et d'outils très nombreux qui ajoutent des détails.

Avenir de XML

- **XML devrait permettre** pour des applications (qui ne posent pas de problèmes cruciaux de performances) de supporter dans un cadre unifié:
- **La présentation des réseaux**
  - Format des données échangées par messages ou par invocations de méthodes.
- **La définition des documents.**
  - Outils de bureautique, de documentation ..
- **La définition des données.**
  - SGBD, logiciels de gestion, Échanges de Données Informatisé (EDI), ...

Le typage XML avec ‘XML schéma’

Introduction
Les structures
Les types de données
Conclusion

Le typage avec les DTD

- **XML est un outil d'avenir** pour l'échange, le stockage et l'affichage des données sur Internet.
- **Un problème majeur de XML 1.0** (dans sa définition de base) concerne **son approche du typage => DTD** 'Document Type Definition'.
- **L'objectif n'est pas atteint** pour des applications informatiques: la définition des types par les DTD est trop orientée documents textuels.
- **Conséquence :** nombreuses propositions pour améliorer le typage en XML.

Objectifs précis

- Structures–

- **Définir la structure et les contenus** des documents.
- **Définir des relations d'héritage.**

- Typage des données –

- **Fournir un ensemble de types primitifs.**
- **Définir un système de typage suffisamment riche.**
- **Distinguer les aspects reliés à la représentation lexicale** des données de ceux gouvernant les données.
- **Permettre de créer des types de données usagers dérivés** de types existants en contraignant certaines propriétés (domaine, précision, longueur, format).

Les structures

Principes généraux des schémas

- **Un schéma XML est un document XML.**
- <?xml version="1.0" encoding="ISO-8859-1"?>
- <xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <!-- Déclaration de deux types d ’éléments -->
- <xsd:element name="nom" type="xsd:string" />
- <xsd:element name= "prenom" type="xsd:string" />
- </xsd:schema>

Exemple d'adresse postale en XML: le document

```
<?xml version="1.0"?>
<Addresse_postale_France pays="France">
   <nom>Mr Jean Dupont</nom>
   <rue>rue Camille Desmoulins</rue>
   <ville>Paris</ville>
   <departement>Seine</departement>
   <code_postal>75600</code_postal>
</Addresse_postale_france >
```

DTD d'une adresse postale

```
<!DOCTYPE une_DTD_adresse
[   <!ELEMENT Addresse_postale_france
        (nom, rue, ville, département, code_postal)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT rue (#PCDATA)>
<!ELEMENT ville (#PCDATA)>
<!ELEMENT département (#PCDATA)>
<!ELEMENT code_postal (#PCDATA)>
<!ATTLIST Addresse_postale_france
        pays  NMTOKEN #FIXED 'France' > ]>
```

Typage d'une adresse postale au moyen d'un schéma XML

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
<xsd:complexType name="Addresse_postale_france" >
        <xsd:sequence>
            <xsd:element name="nom" type="xsd:string" />
            <xsd:element name="rue" type="xsd:string" />
            <xsd:element name="ville" type="xsd:string" />
            <xsd:element name="departement" type="xsd:string" />
            <xsd:element name="code_postal" type="xsd:decimal" />
        </xsd:sequence>
         <xsd:attribute name= "pays" type="xsd:NMTOKEN"
                use="fixed" value="FR"/>
</xsd:complexType>      </xsd:schema>
```

Le langage XML schémas : Les composants primaires

- **Un schéma XML est construit par assemblage de différents composants** (13 sortes de composants rassemblés en différentes catégories).
- **Composants de définition de types**
  - Définition de types **simples (Simple type).**
  - Définition de types **complexes (Complex type).**
- **Composants de déclaration**
  - **Déclaration d'éléments.**
  - **Déclaration d'attributs.**

Déclaration des éléments

- **Un élément XML est déclaré par la balise 'element'** de XML schéma qui a de nombreux attributs.
- Les deux principaux attributs sont:

  - **name** : Le nom de l'élément (de la balise associée).
  - **type** : Le type qui peut être simple ou complexe.

Exemple de base

```
<xsd:element name="code_postal" type="xsd:decimal"/>
```

Déclaration des attributs

- **Un attribut est une valeur nommée et typée associée à un élément.**
- **Le type d'un attribut défini en XML schéma est obligatoirement simple.**

```
<xsd:complexType name="TypeRapport">
   <xsd:attribute name= "Date_creation"
   type="xsd:date"/>
     ……
</xsd:complexType>
<xsd:element name="Rapport" type="TypeRapport"/>
```

Autres attributs

- **L'élément attribute de XML Schema peut avoir deux attributs optionnels** : **use** et **value**.
- **On peut ainsi définir des contraintes** de présence et de valeur.
- Selon ces deux attributs, la valeur peut:

  être obligatoire ou non

  être définie ou non par défaut.
- Exemple: <xsd:attribute name= "Date_peremption" type="xsd:date" use="default" value= "2005-12-31"/>

Valeurs possibles pour use

- **Use = required** : L'attribut doit apparaître et prendre la valeur fixée si elle est définie.
- **Use= prohibited** : L'attribut ne doit pas apparaître.
- **Use = optional** : L'attribut peut apparaître et prendre une valeur quelconque.
- **Use= default** : Si l'attribut à une valeur définie il la prend sinon il prend la valeur par défaut.
- **Use= fixed** : La valeur de l'attribut est obligatoirement la valeur définie.
- **Exemple :** <xsd:attribute name= "Date_creation" type="xsd:date" use="required"/>

Types simples

- **'SimpleType' permet de définir des éléments ou des attributs non structurés** : dérivés d'une chaîne, d'un entier etc
  - **Types simples prédéfinis** au sens de la norme XML Schémas ' datatypes': string, integer, boolean **...**

```
<xsd:element name="code_postal " type="xsd:integer"/>
```

  - **Types simples définis par dérivation d'un autre type simple**, au moyen de l'élément <xsd:simpleType ...>
- Exemple de type simple : dérivation par restriction.

```
<xsd:simpleType name=  "DeuxDecimales">
        <xsd:restriction base="xsd:decimal">
                <xsd:fractionDigits value="2" />
        </xsd:restriction>
</xsd:simpleType>
```

Types complexes

- **Déclarés au moyen de l'élément** <xsd:complexType name="...
- **Ils peuvent contenir d'autres éléments**, des attributs.
- **Exemple**

```
<xsd:complexType name= "TypePrix">
        <xsd:simpleContent>
            <xsd:extension base="DeuxDecimales">
                <xsd:attribute name="Unite" type= "FrancEuro" />
            </xsd:extension>
        </xsd:simpleContent>
</xsd:complexType>
```

- **Trois façons de composer des éléments** dans un type complexe: sequence, choice, all.

Types complexes: Sequence

- Un type **sequence** est défini par une suite de sous-éléments qui doivent être présents dans l'ordre donné.
- Le nombre d'**occurences** de chaque sous-élément est défini par les attributs minOccurs et maxOccurs.

```
<xsd:complexType name= "Commande">
    <xsd:sequence>
        <xsd:element name= "Ad_livraison" type="Addresse"/>
        <xsd:element name= "Ad_facturation" type="Addresse"/>
       <xsd:element name=  "texte" type="xsd:string" minOccurs="1" />
        <xsd:element name="items" type="Items" maxOccurs= "30" />
     </xsd:sequence>
</xsd:complexType>
```

Types complexes: Choice

- Un seul des éléments listés doit être présent.
- Le nombre d'occurences possible est déterminé par les attributs minOccurs et maxOccurs de l'élément.

```
<xsd:complexType name=  "type_temps">
   <xsd:choice >
    <xsd:element name= "Noire"  type="Note" minOccurs="1"
    maxOccurs="1" />
    <xsd:element name= "Croche"  type="Note" minOccurs="2"
    maxOccurs="2" />
   </xsd:choice>
</xsd:complexType>
```

Types complexes: All

- C’est une composition de type ensembliste. Dans un document conforme, les éléments listés doivent être tous présents au plus une fois. Il peuvent apparaître dans n'importe quel ordre.

```
<xsd:complexType name= "Commande">
   <xsd:all>
      <xsd:element name= "Ad_livraison" type="Addresse"/>
      <xsd:element name= "Ad_facturation" type="Addresse"/>
      <xsd:element name= "texte" type="xsd:string" />
      <xsd:element name="items"  type="Items" />
   </xsd:all>
</xsd:complexType>
```

Les types de données XML schéma

Objectifs de la définition des types

- **Fournir des types primitifs** analogues à ceux qui existent en SQL ou en Java.
- **Définir un système de typage suffisamment riche** pour importer/exporter des données d'une base de données.
- **Distinguer les aspects reliés à la représentation lexicale des données** de ceux gouvernant les ensembles de données sous-jacents.
- **Permettre de créer des types de données usagers** dérivés de types existants en contraignant certaines propriétés (domaine, précision, longueur, format).

Système de typage des schémas

Trois composantes:

a) L'ensemble des valeurs du type ('value space')

Ex: type float.

b) L'ensemble des représentations lexicales possibles des valeurs ('lexical space ').

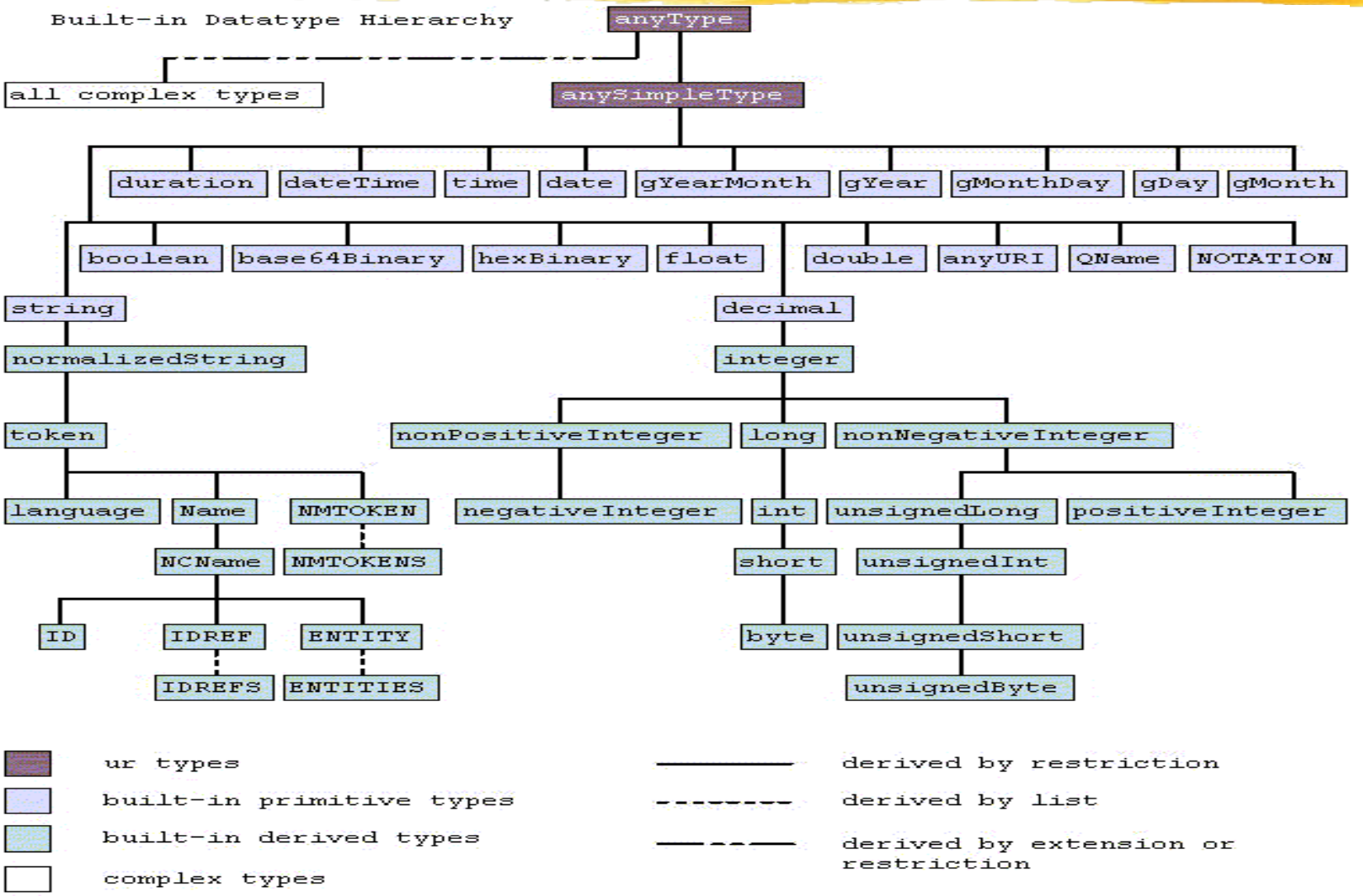
Ex: "10" ou "1.0E1"

c) L'ensemble des facettes (l'ensemble des propriétés) qui définit l'ensemble des valeurs (notion de facette fondamentale et de facette de contrainte).

Ex: Le type float est défini par la norme IEEE 754-1985 (c'est un flottant simple précision sur 32-bit).

On peut dériver des types par contraintes.

Hiérarchie des types prédéfinis



Dérivation de types simples
1- Dérivation par restriction

- **La dérivation par restriction restreint l'ensemble des valeurs** d'un type pré existant.
- La restriction est définie par des contraintes de facettes du type de base: valeur min, valeur max ...
- **Exemple :**

```
<xsd:simpleType name= "ChiffresOctaux">
      <xsd:restriction base="xsd:integer">
            <xsd:minInclusive value="0" />
            <xsd:maxInclusive value= 7" />
      </xsd:restriction>
</xsd:simpleType>
```

Les contraintes de facettes

- lenght : la longueur d'une donnée.
- minLenght: la longueur minimum.
- maxLenght: la longueur maximum.
- pattern: défini par une expression régulière.
- enumeration: un ensemble discret de valeurs.
- whitespace: contraintes de normalisation des chaînes relativement aux espaces (preserve, replace, collapse).
- maxInclusive: une valeur max comprise.
- maxExclusive: une valeur max exclue.
- minInclusive: une valeur min comprise.
- maxInclusive: une valeur min exclue.
- totalDigits: le nombre total de chiffres.
- fractionDigits: le nombre de chiffres dans la partie fractionnaire.

Exemple d’une énumération

```
<xsd:simpleType name= "Mois">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value= "Janvier"/>
                <xsd:enumeration value="Février"/>
                <xsd:enumeration value="Mars"/>
                <!-- ... -->
        </xsd:restriction>
</xsd:simpleType>
```

2 - Dérivation par extension

- Dériver un nouveau type par **extension** consiste à ajouter à un type existant des sous-éléments ou des attributs.
- On obtient inévitablement **un type complexe.**
- **Exemple**

```
<xsd:complexType name= "mesure">
  <xsd:simpleContent><xsd:extension base="xsd:Decimal">
      <xsd:attribute name="unite" type="xsd:NMTOKEN"/>
  </xsd:extension></xsd:simpleContent>
</xsd:complexType>
 <xsd:element name= "temperature" type= "mesure"/>
 <temperature unit="Kelvin">230</temperature>
```

3 - Dérivation par union

- Pour créer un nouveau type on effectue l'union ensembliste de toutes les valeurs possibles de différents types existants.
- Exemple:

```
<xsd:simpleType name="TransportFormatCaracteres">
<xsd:union memberTypes="base64Binary hexBinary"/>
</xsd:simpleType>
```

Conclusion typage avec XML schéma

Un standard très utile en réseaux/systèmes répartis

- **Indispensable comme outil d'interopérabilité en univers réparti.**
  - Entre des applications WEB.
  - Dans des approches objets répartis comme SOAP ('Simple Object Protocol')
  - WSDL ('Web Service Definition Language').
  - Entre des bases de données hétérogènes.
- **Quelques reproches**
  - Les performances.
  - Les imperfections à découvrir dans les choix de conception du système de typage.

Domaines d'utilisation

- Publication d'informations sur le WEB.
- Commerce électronique.
- Gestion de documents traditionnels.
- Assistance à la formulation et à l'optimisation des requêtes en bases de données.
- Transfert de données entre applications en réseaux.
- Contrôle de supervision et acquisition de données.
- Échange d'informations de niveau méta.

Introduction aux services Web

- **Le besoin :** des communications de programme à programme utilisant comme support la toile mondiale.
- **Service toile (service web)**: une application informatique quelconque (plutôt de gestion ou de commerce électronique).
  - modulaire,
  - basée sur la toile,
  - utilisant des services et protocoles standards.
- **Notion de fournisseur de services** sur la toile: (ASP 'Application Service Providers') au moyen des services webs.
- **Applications visées:** commerce électronique puis par extension tout service comportant un accès à des données (température, trafic, …) ou une algorithmique plus ou moins complexe (exemple calcul intensif).

Organisation actuelle des standards

UDDI Services d'annuaire	Ws-Security Services de sécurité	WS-Transaction Services de transaction	Bpel4WS, .. Services de synchronisation

SOAP, WSDL : Interactions de communication

HTTP, SMTP, MOM(JMS) : Transmission effective

SOAP
(‘Simple Object Access Protocol’)

- Le protocole de **communication** de base (définissant la principale interaction de communication).
- **Une approche de mode message** mais aussi un protocole d’appel de procédure distante (**RPC**).
- **Utilise XML** pour représenter les données.
- **Utilise des protocoles applicatifs Internet** pour acheminer ses messages:
  - **HTTP** (pour sa généralité, son mode synchrone).
  - **SMTP** , **MOM** (plutôt pour le mode asynchrone).

WSDL
(‘Web Services Definition Language’)

- **Un standard de description d’interfaces (analogue de IDL).**
- Permet de décharger les utilisateurs des **détails techniques** de réalisation d’un appel.
- **Basé sur XML, XML Schéma**.
- Règles de **sérialisation des données** échangées.

UDDI ‘Universal Description, Discovery, and Integration’

- Le standard **d’annuaire réparti** pour la description des services Web.
- Très orienté **affaires** (ventes, prestations)
- **Accessible au moyen de Soap**.
- Permet d enregistrer des informations variées via la notion de **tmodel** (modèle technique).

Acteurs du domaine

- **Consortiums/organismes**
  - W3C World Wide Web Consortium
  - Oasis
  - WS-I Web Services - Interoperability
  - Nations Unies
  - EbXML
  - Rosetta net : Web services en électronique
- **Editeurs de logiciels**
  - Microsoft, IBM, BEA
- **Logiciel libre**
  - Apache Axis

Exemples de produits

- **Microsoft** .NET
- **IBM** Web Services Architecture (Websphere)
- **SUN** One
- **BEA** Web Services Architecture (Weblogic)
- **Iona**, **CapeClear**, **SilverStream**, **Systinet**

- Logiciel libre **Apache** Axis (WSIF ' Web Services Invocation Framework ')

Bibliographie Normes

- 'XML Schema Part 0: Primer' W3C http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/
- 'XML Schema Part 1: Structures' W3C http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/
- 'XML Schema Part 2: Datatypes' W3C http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/
- XML Schema Requirements http://www.w3.org/TR/1999/NOTE-xml-schema-req-19990215 (Février 1999)
- Introduction aux schémas http://www.w3schools.com/schema
- Les schémas XML Gregory Chazalon, Joséphine Lemoine http://site.voila.fr/xmlschema
- W3C XML Schema, Eric Van Der Vlist, http://www.xml.com/pub/a/2000/11/29/schemas/

XHTML et XSLT

@Francoise Sailhan

Rappel

- Pourquoi a-t-on intérêt à séparer le fond et la forme d'un document?
  - Pour se focaliser sur les données soit sur la forme
    - L'ingénierie des connaissances et la publications sont deux corps de métier différents
  - Pour restructurer et présenter les données de différentes manières en fonction
    - du terminal (téléphone, ordinateur)
    - de la sortie (textuelle, graphique, sonore, mixte)
    - de la charte graphique (sobre, colorée, délirante)

Mise en forme

- Ce que nous avons vu
  - HTML, XHTML, XML
  - Vérification du respect des règles d'écriture et d'enchâssements des éléments
    - DTD, Schémas XML
- Mise en forme et adaptation aux formats de visualisation
  - CSS (Cascading Style Sheet)
  - XSL (*eXtensive Stylesheet Langage* ou langage extensible de feuille de style)

XSL

- XSL recouvre 3 langages
  - **XPath** : un langage pour naviguer dans un document XML
    - moyen d'accéder à un nœud quelconque de l'arborescence XML
  - **XSLT** (*eXtensible Stylesheet Langage Transformation)* : plus que des feuilles de style, un langage de transformation
    - Langage de feuille de style (une transformation est appelée feuille de style)
    - très puissant manipulateur d'éléments permettant de transformer un document XML source (arbre source) en un autre (arbre résultat)
      - Permet de créer des sommaires, tables de matières, de modifier un document à partir de critères d'extraction, de tri, ect…
  - **XSL-FO** (*eXtensible Stylesheet Langage - Formating Objects)* : un langage pour formater
    - permet le contrôle de la mise en page finale de la transformation
    - est destiné à la production de contenus au format PDF
- XSLT nécessite un document XML bien formé

Structure d'un document XSL

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  (...)
  </xsl:stylesheet>
```

- xsl:stylesheet : élément racine du document

Exemple – document XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="bouteille1.xsl"?
>
<bouteille>
  <marque>Cristaline</marque>
  <composition>calcium 71mg/l, magnésium 5,5mg/l,
chlorure 20mg/l, nitrate 1mg/l, traces de
fer.</composition>
  <source>
    <ville>St-Cyr la Source</ville>
    <departement>Loiret</departement>
  </source>
  <code_barre>3274080005003</code_barre>
</bouteille>
```

Exemple – feuille XSL associée

```
<?xml version="1.0" encoding="ISO-8859-1"?>
   <xsl:stylesheet version="1.0«  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
     <html>
       <head>
        <title>Exemple de sortie HTML</title>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" /
   >
       </head>
       <body>
        <h1>Bouteille de marque <xsl:value-of select="bouteille/marque" /></h1>
        <h2>Composition:</h2>
        <p><xsl:value-of select="bouteille/composition" /></p>
        <h2>Lieu d'origine:</h2>
        <p>Ville de <b><xsl:value-of select="bouteille/source/ville" /></b>, dans
   le département <b><xsl:value-of
   select="bouteille/source/departement" /></b></p>
        <h2>Autres informations</h2>
        <ul>
          <li>Contenance: <xsl:value-of select="bouteille/contenance" /></li>
          <li>pH: <xsl:value-of select="bouteille/ph" /></li>
        </ul>
       </body> </html></xsl:template></xsl:stylesheet>
```

XPATH – Sélection d'un nœud (1)

- Se fonde sur la structure arborescente du document XML pour faire référence à des éléments et/ou des attributs
- Syntaxe pour la sélection d'un nœud : <xsl:value-of select="nom_element" />
- Expressions associées à la sélection :
  - / définie le chemin d'accès aux éléments à sélectionner, et donc leur parenté
    - Exemple : section/paragraphe
      - sélectionne les éléments section du nœud courant et pour chaque élément section, sélectionne les éléments paragraphe qu'il contient
      - sélectionne les petits-fils paragraphe du nœud courant qui ont pour père un nœud section

XPATH – Sélection d'un nœud (1)

- * peut remplacer le nom de n'importe quel nœud dans une expression
  - Exemple : */paragraphe sélectionne tous les petits-fils paragraphe quel que soit leur père
- // applique la recherche aux descendants et non pas seulement aux fils directs
  - Exemple : section//paragraphe sélectionne tous les éléments paragraphe descendant d'un élément section fils direct du nœud courant
- . sélectionne le nœud courant
  - Exemple : .//paragraphe sélectionne tous les descendants paragraphe du nœud courant.
- .. sélectionne le parent du nœud courant
  - Exemple, ../paragraphe sélectionne tous les nœuds paragraphe frères du nœud courant.

XPATH – Sélection d'un nœud (2)

- **Appel de fonctions**
  - comment() sélectionne tous les nœuds commentaires fils du nœud courant
  - text() sélectionne tous les nœuds fils du nœud courant, ne contenant que du texte
  - node() sélectionne tous les nœuds fils du nœud courant
  - id("UnIdentifiant") sélectionne l'élément, normalement unique, qui a un attribut attr de type ID valant "UnIdentifiant".

XPATH – Sélection d'un nœud (3)

- La **navigation** dans les branches de l'arborescence du document XML se fait avec une expression de la forme :

  Element[Expression]

  prédicat

- Exemples (où i est un nombre entier)
  - elt[i] désigne le i-ème descendant direct d'un même parent ayant le nom indiqué.
    - Exemple : paragraphe[3]
      - désigne le *3*ème enfant de l'élément courant, portant le nom paragraphe.
      - *Attention*, la numérotation commence à 1 et non à 0!

XPATH – Sélection d'un nœud (4)

- elt[position()>i] sélectionne tous les éléments précédés d'au moins i éléments de même nom comme descendants du même parent
  - Exemple : paragraphe[position()>5] sélectionne tous les éléments paragraphe dont le numéro d'ordre est strictement supérieur à 5
- elt[position() mod 2=1] sélectionne tout élément qui est un descendant impair
- elt[souselt] sélectionne tout élément elt qui a au moins un descendant souselt
- elt[first-of-any()] sélectionne le premier élément elt fils de l'élément courant
- elt[last-of-any()] sélectionne le dernier élément elt fils de l'élément courant

XPATH – Sélection d'un nœud (5)

- elt[first-of-type()] sélectionne l'élément elt fils de l'élément courant, s'il est premier de son type.
  - Exemple : elt2[first-of-type()]
    - l'élément elt peut contenir des nœuds de type texte elt1 et elt2 dans n'importe quel ordre
    - Deux cas se présentent :
      - elt commence par au moins un élément elt1, avec *éventuellement* un élément elt2 ensuite,
      - elt commence par un élément elt2.
    - L'expression ne sélectionne le premier élément elt2 que dans le second cas, où il n'est précédé par aucun élément de même type.

XPATH – Sélection d'un nœud (6)

- elt[last-of-type()] sélectionne de même l'élément elt fils de l'élément courant, s'il est le dernier de son type.
  - Exemple : section/paragraphe[last-of-type() and first-of-type()]
    - sélectionne les éléments paragraphe fils uniques dont le père est un élément section ;
    - section/paragraphe[last-of-any() and first-of-any()] sélectionne les éléments paragraphe dont le père est un élément section qui ne contient qu'un seul élément paragraphe.
- ancestor() sélectionne un ancêtre du nœud courant. Elle reçoit en argument une expression de sélection et recherche le premier ancêtre du nom correspondant à la sélection
  - Exemple : ancestor(chapitre)/titre sélectionne l'élément titre du chapitre contenant l'élément courant

XPATH – Sélection d'un nœud (7)

- **Sélection d'attributs**
  - Les attributs d'un élément sont sélectionnés en faisant précéder leur nom par le caractère @
  - Règles relatives à la sélection des éléments s'appliquent également aux attributs :
- Exemple
  - section[@titre] sélectionne les éléments section qui ont un attribut titre
  - section[@titre="Introduction"] sélectionne les éléments section dont l'attribut titre a pour valeur Introduction
  - Affichage du contenu de l'attribut en le faisant précéder du caractère /
    - Exemple : <xsl:value-of select="paragraphe/@titre" />
    - Affichage du titre de l'élément paragraphe fils de l'élément courant
    - Si rien n'est précisé, par défaut il s'agit du premier élément paragraphe fils

XPATH – Sélection d'un noeud (8)

- Opérateur logiques
  - not()
  - and
  - or
- Opérateurs + - * div = != < <= > >= mod |
- Exemple : section[not(@titre)]
  - sélectionne les éléments section qui n'ont pas d'attribut titre

XPATH - chemins

- Un chemin de localisation est composé de trois parties :
  - un *axe*, définissant le sens de la relation entre le nœud courant et le jeu de nœuds à localiser
  - un nœud spécifiant le type de nœud à localiser
  - 0 à n *prédicats* permettant d'affiner la recherche sur le jeu de nœuds à récupérer
  - Exemple : chemin child::section[position()=1]
    - child est le nom de l'axe
    - section le type de nœud à localiser (élément ou attribut)
    - [position()=1] est un prédicat

XPATH – chemins : les axes

- Le moi
  - self : contient seulement le nœud contextuel
  - attribute : contient les attributs du nœud contextuel ; l'axe est vide quand le nœud n'est pas un élément
- Enfants et descendants
  - child : contient les enfants directs du nœud contextuel
  - descendant :
    - contient les descendants du nœud contextuel.
    - Un descendant peut être un enfant, un petit-enfant...
  - descendant-or-self : contient le nœud contextuel et ses descendants.
- Parents et ancêtres
  - parent : contient le parent du nœud contextuel, s'il y en a un.
  - ancestor : contient les ancêtres du nœud contextuel (père, le père de son père...)
  - ancestor-or-self : contient le nœud contextuel et ses ancêtres. Cet axe contient toujours le nœud racine.

XPATH – chemins : les axes

- Les pairs
  - following-sibling :
    - contient tous les nœuds cibles du nœud contextuel
    - Si ce nœud est un attribut ou un espace de noms, la cible suivante est vide
  - preceding-sibling :
    - contient tous les prédécesseurs du nœud contextuel
    - si le nœud contextuel est un attribut ou un espace de noms, la cible précédente est vide.
  - following : contient tous les nœuds du même nom que le nœud contextuel situés après le nœud contextuel dans l'ordre du document, à l'exclusion de tout descendant, des attributs et des espaces de noms.
  - preceding : contient tous les nœuds du même nom que le nœud contextuel situés avant lui dans l'ordre du document, à l'exclusion de tout descendant, des attributs et des espaces de noms.
- namespace : contient tous les nœuds des espaces de noms du nœud contextuel ; l'axe est vide quand le nœud contextuel n'est pas un élément.

XPATH – chemins : prédicats

- Le contenu d'un prédicat est une prédiction
- Le résultat de chaque expression est un booléen
- Exemple, section[3] équivaut à section[position()=3]

XPATH - chemins

- 2 types de chemins de localisation
  - Chemin absolu : commence toujours par le signe / indiquant la racine du document XML
  - Chemin relatif : le nœud de départ est le nœud contextuel courant
- 2 types de syntaxe de composition d'un chemin de localisation
  - syntaxe abrégée
    - recoupe la « syntaxe de base » qui a déjà été présentée
    - permet d'obtenir des expressions du type para[@type="avertissement"][5], qui sélectionne le cinquième enfant de l'élément para, parmi ceux qui ont un attribut type ayant la valeur avertissement.
  - Syntaxe non abrégée : child::*para* - child::* - child::text() - child::node() - attribute::*name* - attribute::* - descendant::*para* - ancestor::*div* - ancestor-or-self::*div* - descendant-or-self::*para* - self::*para* - child::*chapitre*/descendant::*para* - child::*/child::*para* - /child:: - /descendant::*para* - /descendant::*olist*/child::*item* - child::*para*[position()=1] - child::*para*[position()=last()] - child::*para*[position()=last()-1] - child::*para*[position()1] - following-sibling::*para*[position()=1]

XPATH – Fonctions de base

- Les fonctions manipulent 4 catégories d'objets :
  - Nœuds
  - Chaînes de caractères
  - Booléens Nombres
- Chaque fonction peut avoir zéro ou plusieurs arguments (on utilise alors le caractère ? Pour définir un argument optionnel)

XPATH – Fonctions de base

- **Manipulation de nœuds**
- *Fonctions retournant un nombre :*
  - last() : retourne un nombre égal à l'index du dernier nœud dans le contexte courant.
  - position() : retourne un nombre égal à la position du nœud dans le contexte courant.
- *Fonction retournant un jeu de nœuds* : id(objet), permet de sélectionner les éléments par leur identifiant.

XPATH – Fonctions de base

- **Fonctions de manipulation de chaînes de caractères**
- string(nœud?) : convertit un objet en chaîne de caractères selon les règles suivantes :
  - Un ensemble de nœuds est converti en chaîne de caractères en retournant la valeur textuelle du premier nœud de l'ensemble dans l'ordre du document.
  - Si l'ensemble des nœuds est vide, une chaîne vide est retournée.
  - un nombre est converti en chaîne, la valeur booléenne false/true est convertie en chaîne de caractères "false"/"true".
- concat(chaine1, chaine2, chaine*) : retourne une chaîne résultant de la compilation des arguments
- string-length(chaine?) : retourne le nombre de caractères de la chaîne. Si l'argument est omis, la valeur retournée est la longueur de la valeur textuelle du nœud courant

XPATH – Fonctions de base

- **Fonctions de manipulation de booléens**
- not(), true(), false()
- lang(chaine) :
  - teste l'argument chaine par rapport à l'attribut xml:lang du nœud contextuel ou de son plus proche ancêtre dans le cas où le nœud contextuel ne contient pas d'attribut de ce type
  - retourne true si l'argument est bien la langue utilisée ou si la langue utilisée est un sous-langage de l'argument (par exemple, en//us).
  - sinon elle retourne false
- **Fonctions de manipulation de nombres**
- floor(nombre) : retourne le plus grand entier inférieur à l'argument passé à la fonction
- ceiling(nombre) : retourne le plus petit entier supérieur à l'argument passé à la fonction
- round(nombre) : retourne l'entier le plus proche de l'argument passé à la fonction

XSLT

- XSLT est utilisé pour transformer des documents XML en un document se trouvant dans un autre format
  - Utilisé typiquement pour transformer dans un format reconnu par le navigateur : XML → XML ou XML → HTML ou XML → XHML
- Se base sur XPath pour naviguer (trouver les nœuds) dans le document XML
- XSLT dispose de 2 fichiers en entrée
  - Document XML contenant les données
  - Document XSL (.xsl)
    - contenant le framework dans lequel les données doivent être insérées
    - Les commandes XSLT permettant cette insertion

XSLT : structure

- Au niveau du prologe du fichier .xls : 2 façons de déclarer la racine

```
<?xml version="1.0"?>
  <xsl:stylesheet version="1.0"xmlns:xsl="http://www.w3.org/1999/
                  XSL/Transform">
```

Ou

```
<?xml version="1.0"?>
  <xsl:transform version="1.0"xmlns:xsl="http://www.w3.org/1999/
                  XSL/Transform">
```

- Contient un ou plusieurs templates :

  <xsl:template match="/"> ... </xsl:template>

- Se termine avec :

  </xsl:stylesheet> ou </xsl:transform>

Lien avec le .xsl dans le document .xml

- Prologue du fichier .xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <?xml-stylesheet type="text/xsl" href="monfichier.xsl"?
  >
```

XSLT – Structure : prologue et racine

- Prologue

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

  <xsl:stylesheet id="id" version="nombre" xmlns:pre="URI"> (...) </xsl:stylesheet>
  - id : identifiant unique de la feuille de style
  - version : numéro de version de la feuille de style XSLT
  - xmlns:pre : définition de l'espace de noms
    - pre : préfixe utilisé dans la feuille de style pour faire référence à l'URI de l'espace nominal
- Exemples :

  <xsl:stylesheet version="1.0" xmlns:xsl="uri:xsl"> (...) </xsl:stylesheet>

XSLT – Structure : options de l'arbre de sortie

- **<xsl:output>**
  - premier enfant de <xsl:stylesheet>
  - spécifie des options concernant l'arbre de sortie

<xsl:output method="xml | html | text" version="nmtoken" encoding="chaine" omit-xml-declaration="yes | no" standalone="yes | no" doctype-public="chaine" doctype-system="chaine" cdata-section-elements="elt" indent="yes | no" media-type="chaine" />

  - method identifie la méthode de transformation. Si elle est égale à text, aucune mise en forme n'est effectuée
  - version identifie la version de la méthode de sortie (xml1.0, html4.01...).
  - encoding indique la version du jeu de caractères à utiliser pour la sortie
  - omit-xml-declaration indique au processeur XSLT s'il doit ajouter ou non une déclaration XML
  - standalone indique au processeur XSLT s'il doit créer un arbre de sortie avec ou sans déclaration de type de document.
  - doctype-public indique l'identifiant public utilisé par la DTD associée à la transformation
  - doctype-system indique l'identifiant system utilisé par la DTD associée à la transformation
  - cdata-section-elements indique les éléments dont le contenu doit être traité lors de la transformation via une section CDATA.
  - indent présente la transformation sous forme d'arbre dans le cas où la valeur de cet attribut est égale à yes
  - media-type indique le type MIME des données résultantes de la transformation
- Exemple : <xsl:output method="html" version="html4.01" encoding="ISO-8859-1" doctype-public="-//W3C//DTD HTML 4.01//EN" doctype-system="http://www.w3.org/TR/html4/strict.dtd" />
  - fichier de sortie au format HTML 4.01, conforme à la DTD publique de l'HTML du W3C, avec le jeu de caractères l'ISO-8859-1
  - Le résultat en sortie est un fichier HTML commençant par
    <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
  - ... et possédant dans son <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />

XSLT – Structure : modèle

- **<xsl:template>**
  - définit un modèle à appliquer à un nœud et à un contexte spécifique

<xsl:template name="nommodele" match="expression" mode="modemodele"> </xsl:template>

    - name : nom associé au modèle
    - match :
      - indique quel jeu de nœuds sera affecté par le modèle
      - peut comprendre un test d'existence d'attribut, le caractère | indiquant que le modèle s'applique à un élément ou à un autre, ainsi que tout élément permettant de définir un jeu d'attributs.
      - mode permet à un élément d'avoir plusieurs modèles, chacun générant une sortie différente.

XSLT – Structure : valeur d'un nœud

- **<xsl:value-of>**
  - Sélectionne la valeur d'un nœud et l'insère dans la transformation
  - Ce nœud est évalué en fonction d'une expression (un élément, un attribut ou tout autre nœud contenant une valeur)

<xsl:value-of select="expression" disable-output-escaping="yes | no" />

  - La valeur de select est évaluée et c'est cette évaluation qui sera insérée dans la transformation
  - disable-output-escaping agit sur la transformation des caractères.
    - Si sa valeur est yes, la chaîne < est insérée dans la transformation en tant que signe <
    - Sinon, la chaîne < est insérée telle quelle dans la transformation

XSLT – ajouts d'éléments et attributs

- **<xsl:element>**
  - insère un nouvel élément
  - Le nom de l'élément est donné par l'attribut name

<xsl:element name="nomelement" use-attribute-sets="jeuattr"> </xsl:element>

  - name : nom de l'élément à créer
  - use-attribute-sets : jeu d'attributs à associer à l'élément créé
- Exemple : <xsl:element name="p"><xsl:value-of select="texte" /></xsl:element>
  - crée dans le fichier HTML un élément de paragraphe renfermant le contenu de l'élément texte du document XML

XSLT – ajouts d'éléments et attributs

- **<xsl:attribute>**
  - définit un attribut et l'ajoute à l'élément résultat de la transformation
- <xsl:attribute name="nom">valeur</xsl:attribute>
  - name : nom de l'attribut à ajouter dans le contexte courant
  - valeur : valeur à lui donner
- Exemple : <image><xsl:attribute name="src">test.gif</xsl:attribute></image>
  - ajoute à l'élément image l'attribut src et lui affecte la valeur test.gif
  - Résultat : <image src="test.gif"></image>

XSLT – boucles

- **<xsl:for-each> :** crée une boucle dans laquelle sont appliquées des transformations

<xsl:for-each select="expressionXPath">texte à insérer et rêgles à appliquer</xsl:for-each>

  - select : jeu de nœuds devant être parcouru par la boucle
  - Exemple d'utilisation :

```
<ul>
    <xsl:for-each select="item">
        <li><xsl:value-of select="." /></li>
    </xsl:for-each>
</ul>
```

XSLT – boucles

- Autre exemple avec un filtrage additionnel permettant de réduire en ajoutant un critêre poru sélectionner la valeur de l'attribut

```
<ul>
  <xsl:for-each select="//book">
    <li>
    <xsl:value-of
      select="title[../author='Victor Hugo']"/>
    </li>
    </xsl:for-each>
</ul>
```

- Exemple sélectionnant le livre ecrits par Victor Hugo
  - Filtre appliqué : `<xsl:value-of select="title[../author='Terry Pratchett']"/>`
  - author est un pair de title, donc à partir de title on doit remonter au parent puis redescendre a author
  - Ce filter nécessite de simple ' dans de doubles " : `"title[../author='Victor Hugo']"/>`
- Les filtre disposibles sont : = != < >

XSLT – boucles

- **<xsl:sort>**
  - Effectue un tri sur un jeu de nœuds
  - Est placé soit dans un <xsl:for-each> soit dans un <xsl:apply-templates>
  - C'est un élément vide qui peut être appelé plusieurs fois pour effectuer un tri multicritères; chaque appel provoquant un tri sur un champ spécifique

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

  - select : spécifie un nœud comme clé de tri
  - data-type : type des données à trier
  - order : ordre de tri prenant les valeurs ascending ou descending
  - lang : jeu de caractères utiliser pour le tri ; par défaut, il est déterminé en fonction des paramètres système
  - case-order : indique si le tri a lieu sur les majuscules (prend la valeur upper-first) ou minuscules (valeur lower-first) en premier ou

- Exemple :

```
<ul>
    <xsl:for-each select="livre">
      <xsl:sort select="auteur" order="descending" />
            <li><b><xsl:value-of select="auteur" /></b><br /><xsl:value-of select="titre" /></li>
    </xsl:for-each>
</ul>
```

- La liste des livres est classée dans l'ordre alphabétique décroissant des noms d'auteur.

XSLT – boucles

- **<xsl:number> :** insére un nombre formaté pouvant servir de compteur
- <xsl:number level="single | multiple | any" count="nœud" from="nœud" value="expression" format="chaine" lang="nmtoken" grouping-separator="car" grouping-size="nombre" />
  - level : niveaux à sélectionner pour le comptage
    - = valeur single : ne compter que la position du nœud par rapport à ses frères
    - = valeur multiple : permet de compter la position du nœud par rapport à ses ancêtres (par exemple de la forme 1.3.4) ; enfin any renvoie la position du nœud par rapport à l'ensemble des nœuds de même nom dans tout le document.
  - count indique quels nœuds doivent être comptés dans les niveaux sélectionnés ; dans le cas où cet attribut n'est pas défini, les nœuds comptés sont ceux ayant le même type que celui du nœud courant.
  - from identifie le nœud à partir duquel le comptage commence.
  - value indique l'expression correspondant à la valeur du compteur ; si cet attribut n'est pas défini, le nombre inséré correspond à la position du nœud (position()).
  - format spécifie le format de l'affichage du nombre ; cela peut être un chiffre, un caractère (a-z, A-Z) et comprendre un caractère de séparation tel que le point (.), le trait d'union (-) ou autre. Les formats possibles sont "1", "01", "a", "A", "i", "I".
  - lang spécifie le jeu de caractères à utiliser ; par défaut, il est déterminé en fonction des paramètres du système.
  - grouping-separator identifie le caractère permettant de définir la séparation entre les centaines et les milliers.
  - grouping-size spécifie le nombre de caractères formant un groupe de chiffres dans un nombre long ; le plus souvent la valeur de cet attribut est 3. Ce dernier attribut fonctionne avec...
  - ...grouping-separator ; si l'un des deux manque, ils sont ignorés.
- Exemple d'utilisation :

```
<ul>
    <xsl:for-each select="livre">
      <xsl:sort select="auteur" />
      <xsl:number level="any" from="/" format="1." />
      <li><b><xsl:value-of select="auteur" /></b><br /><xsl:value-of select="titre" /></li>
    </xsl:for-each>
    </ul>
```

XSLT : tests

- **<xsl:if>** permet
  - la fragmentation du modèle dans certaines conditions
  - de tester la présence ou la valeur d'un attribut, d'un élément,
  - de savoir si un élément est bien le fils d'un autre ect…
- <xsl:if test="condition">action</xsl:if>
  - test : = 1 ou 0 suivant le résultat de la condition (vrai ou faux)
  - action : action devant être effectuée (texte à afficher, second test, gestion de chaîne...)
-
```
<xsl:if test="livre">
  <ul>
    <xsl:for-each select="livre">
      <li>
        <b><xsl:value-of select="auteur" /><br /></b>
        <xsl:value-of select="titre" />.<xsl:if test="@langue='français'">livre en français </xsl:if>
      </li>
    </xsl:for-each>
  </ul>
</xsl:if>
```
- Dans le code précédent, si l'attribut langue de l'élément livre vaut français, le processeur ajoutera au fichier de sortie la phrase "Ce livre est en français". Il ne se passe rien si ce n'est pas le cas.

XSLT : tests

- **<xsl:choose>** définit une liste de choix et affecte à chaque choix une transformation différente

```
<xsl:choose>
    <xsl:when test="une condition">
        ... du code ...
    </xsl:when>
    <xsl:otherwise>
        ... du code ...
    </xsl:otherwise>
</xsl:choose>
```

  - <xsl:when> définit chaque choix
  - <xsl:otherwise> définit un traitement par défaut
- Exemple

```
<ul>
  <xsl:for-each select="livre">
    <li>
      <b><xsl:value-of select="auteur" /><br /></b>
      <xsl:value-of select="titre" />
      <xsl:choose>
        <xsl:when test="@langue='francais'">Ce livre est en français.</xsl:when>
        <xsl:when test="@langue='anglais'">Ce livre est en anglais.</xsl:when>
        <xsl:otherwise>Ce livre est dans une langue non répertoriée.</xsl:otherwise>
      </xsl:choose>
    </li>
  </xsl:for-each>
</ul>
```

XSLT : variables et paramètres

- **<xsl:variable>**
  - sert à créer les variables
  - possède les attributs suivants :
    - name : nom de la variable (attribut obligatoire)
    - select : expression XPath qui spécifie la valeur de la variable
- Exemple :

```
<xsl:variable name="nombre_livres" select="255" />
<xsl:variable name="auteur" select="'Victor Hugo'" />
<xsl:variable name="nombre_pages" select="livre/tome/@page" />
```

- La portée d'une variable est limitée aux frères et à leurs descendants
  - si une variable est déclarée dans une boucle xsl:for-each ou un élément xsl:choose ou xsl:if, on ne peut s'en servir en-dehors de cet élément
- Une variable est appelée en étant précédée du caractère $:

  <xsl:value-of select="**$nombre_pages**" />.

Modularisation

- Objectif de la modularisation : casser un programme complexe en plusieurs parties

- En utilisant des fonctions
  - En XSL, on utilise l'équivalant : xsl:apply-templates

XSLT : variables et paramètres

- **<xsl:call-template>**
  - peut être appelé indépendamment d'une sélection d'un nœud
  - Pour cela, il faut renseigner l'attribut name, et l'appeler à l'aide de l'élément <xsl:call-template>
- Exemple

```
<xsl:template name="separateur">
 <hr />
      <img src="ligne.gif" alt="séparation" width="150" height="2" />
 <hr />
</xsl:template>
```

- Il suffit alors de l'appeler avec <xsl:call-template name="separateur"/>.

XSLT : variables et paramètres

- Créer et passer des valeurs au modèle :
  - **<xsl:param> permet de créer un paramètre**
  - **<xsl:with-param> permet de passer une élément au modèle**
- Tout deux ont 2 attributs :
  - name : donne un nom au paramètre (attribut obligatoire)
  - select : expression XPath donnant une valeur par défaut au paramètre (facultatif)
- Exemple d'un template évaluant le résultat d'une expression polynômiale :

```
<xsl:template name="polynome">
  <xsl:param name="variable_x" />
  <xsl:value-of select="2*$variable_x*$variable_x+(-5)*$variable_x+2" />
</xsl:template>
```

- Appel effectué en passant diverses valeurs pour le paramètre variable_x :
-

```
<xsl:call-template name="polynome">
  <xsl:with-param name="variable_x" select="3.4" />
</xsl:call-template>
```

- ... permet d'afficher le résultat de 2*3.4^2-5*3.4+2.
- On remarque que la soustraction d'un nombre se fait par addition du nombre (négatif) opposé ;
- la division se fait l'opérateur div et non pas /

XSLT – tout cela ensemble…

- Dans un fichier xsl :

```
<xsl:template match="/">
        <html><body>
      <h1><xsl:value-of select="message"/></h1>
        </body></html>
  </xsl:template>
```

  - <xsl:template match="/"> sélectionne la racine
  - <html><body> <h1> … </h1> </body></html> est écris tel quel dans le fichier en sorte
  - Le contenu de message (*Howdy!*) est écris
- Résultat (fichier en sortie):

```
<html><body>
   <h1>Howdy!</h1>
</body></html>
```

Comment cet exemple marche?

- Le document XML est lu et est sauvegardé sous la forme d'un arbre
- Le template <xsl:template match="/"> est utilisé pour sélectionner l'arbre dans son entier
- Les rêgles à l'intérieur du template sont appliquées au noeuds, ce qui change structure de l'arvre XML
- Les parties de l'arbre XML qui ne match pas restent inchangées
- Après que le template ait été appliqué, l'arbre est écris en tant que document texte

Quand utiliser XSLT

- Avec un programme adapté (par exemple Xerces) , XSLT peut être utilisé pour lire et écrire des fichiers
- Un serveur peut utiliser XSLT pour changer des documents XML files en fichiers HTML avant un envoie vers le client
- Un navigateur *moderne* peut utiliser XSLT pour changer des documents XML en fichiers HTML au niveau du client

Navigateurs

- Navigateur supportant XML, XSLT, XPath
  - Mozilla Firefox v3
  - Internet Explorer v6
  - Chrome v1
  - Opera v9
- Navigateur supportant XML, XSLT
  - Safari v3

Avantages et inconvénients

- Avantages
  - Faire un pont entre un document XML et la présentation de ce document
- Désavantage
  - Performances (consommateur de mémoire)

Bibliographie

- http://www.w3schools.com/
- http://www.gchagnon.fr/cours/xml/index.html