XML

eXtensible Markup Language

Anas ABOU EL KALAM

Plan

- Introduction au langage XML
- Un premier exemple
- Règle de syntaxe XML
- Mise en œuvre d'une DTD

Plan

- Introduction au langage XML
- Un premier exemple
- Règle de syntaxe XML
- Mise en œuvre d'une DTD

XML, pourquoi ?

- Limites HTML
  - nombre limité de balises,
  - 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....*
  - **Pas de type de document** (pas de structuration) prédéfinie permettant de définir abstraitement une structure de document
- est il est possible d'échanger des documents sans pour autant influer sur la forme de ceux-ci ?
  - Cela permettrai ainsi de rendre simple l'adaptation d'un contenu à un navigateur ou bien à n'importe quel périphérique d'affichage

XML, pourquoi ?

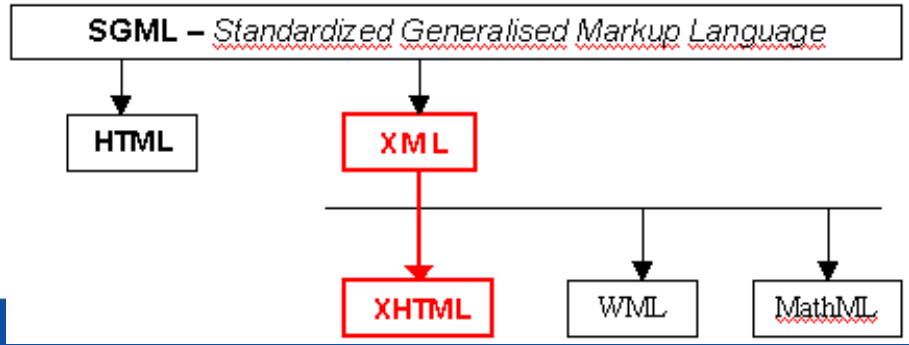
- échanger, via le Web, des données stockées dans des bases de données, …
- W3C ➔ Solution
  - Standard
  - Palliant problèmes de HTML

- Séparation structure // présentation

  - accepté et utilisé de tous
  - pouvoir permettre de transférer des données sur le Web, sans devoir refaire un protocole de communication, et sans forcément avoir de grosses modifications à apporter aux outils utilisés

SGML, HTML, XML, XHTML, … ?

- SGML (*Standard Generalized ML*) est un langage normalisé de balises pour décrire structure et contenu de ≠ types de documents électroniques
- XML & HTML issus de SGML ==> balises, plateforme, mode … Mais
  - XML décrit, structure, échange des données tandis que le Html ne fait qu'afficher des données
  - XML est extensible (créer ses propre balises), HTML est figé
- XHTML est le successeur du Html .... mais aussi un des "enfants" de XML
  - ... pour faire un peu le ménage dans les dérives du Html, le W3C a conçu le XHTML qui n'est en fait qu'une reformulation du HTML 4.0 selon la syntaxe et les règles du XML



XML, c'est quoi ?

- Langage très général utilisant des Tags
- n'a pas pour but de replacer HTML, mais plutôt d'être complémentaire
  - XML n'est pas un langage de présentation de document en tant que tel
- meta-langage : langage permettant de définir un nouveau langage (de nouvelles balises).
  - on peut difficilement exploiter XML tel quel : il faut en spécifier un sous-ensemble sur lequel on pourra travailler.
  - Pour exploiter un sous ensemble de tag XML, on peut utiliser une DTD et des feuilles de styles

- document XML est constitué de plusieurs fichiers et notamment,
  - un fichier de **données**,
  - un fichier de **validation** et
  - une feuille de **styles**

Plan

- Introduction au langage XML
- Un premier exemple
- Règle de syntaxe XML
- Mise en œuvre d'une DTD

Exemple simple

- Exemple,
  - plans de formation d'un grand centre de formation
- Objectif,
  - présentation facile des plans de cours,
  - MAJ / Changement mise en page facile des plans de cours
    - sans forcément devoir retoucher l'intégralité des plans de cours

- XML s'est donc imposé à nous,
- nous avons définit notre propre langage de présentation de plan de cours : appelons le FML (**Formation Markup Language**).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml:stylesheet type="text/xsl" href="Formation.xsl" ?>
<!DOCTYPE FORMATION SYSTEM "Formation.dtd">
<FORMATION>
    <TITRE> Technos C/S (Niv 1 - Fr)</TITRE>
    <DUREE>5</DUREE>
    <PREREQUIS>  Connaissances réseau vivement conseillée     </PREREQUIS>
    <OBJECTIF>    Comprendre les différentes technologies à base de C/S   </OBJECTIF>
    <CONTENU>
        <CHAPITRE>
            <TITRE>      Introuction  </TITRE>
                <SECTION> Les architectures C/S                    </SECTION>
                <SECTION> Communication par messages en mode asynchrone </SECTION>
            <!-- La suite du chapitre -->
        </CHAPITRE>
        <CHAPITRE>
            <TITRE>      Middleware            </TITRE>
                <SECTION> Introduction                          </SECTION>
                <!-- La suite du chapitre -->
        </CHAPITRE>
          <!-- Les autres chapitres -->
    </CONTENU>
    <ANIMATEUR>  Toto  </ANIMATEUR>
    <ANIMATEUR>  titi       </ANIMATEUR>
</FORMATION>
```

Remarques

- **<?xml version="1.0" encoding="ISO-8859-1"?>**
  - … ce qui suit est un document XML selon sa version 1.0.
  - balise sans fermeture car cette balise n'est pas encore du XML
  - indique à l'interpréteur XML [Parser] le jeu de caractères à utiliser
    - *"ISO-8859-1" ==> accepter la plupart des lettres avec des accents.*
    - *UTF-8 ou UTF-16 pour les anglo-saxons, etc.*
- **<racine>** **(*FORMATION dans notre cas*)**
  - L'élément racine est indispensable
  - Vous pouvez utiliser n'importe quel nom à l'intérieur de cette balise
- ... suite du document XML …
  - Votre document XML proprement dit, qui respectera bien entendu scrupuleusement la syntaxe du XML ("bien formé").
- **</racine>**
  - Document XML se termine obligatoirement à la fermeture de la balise de racine.

Remarques

- Une formation (tag <FORMATION>) est constituée d'un titre (tag TITRE), puis d'une durée (tag DUREE), et ainsi de suite.
  - tous les noms de tags sont en français.
  - C'est donc clair,
  - ce n'est pas du prédéfini.
    - *C'est moi qui ai décidé d'appeler ainsi mes tags*

Les 2 premières lignes

- "*formation.**dtd***"
  - définir la grammaire (comment correctement utiliser les tags)
- " *formation.**xsl***".
  - permettre une mise en page d'un fichier de données.

Grammaire

- Ex : HTML
  - un tag <TD> (Table Data) ne peut pas s'utiliser n'importe ou. Une cellule de données TD doit être contenue dans une ligne de tableau <TR> (Table Row) qui doit elle même être contenue dans un tag <TABLE>.
- DTD / Schéma : définir
  - structure d'un document XML,
  - relation entre éléments,
  - type données,
  - contraintes de contenu

Définition d'une grammaire

- deux principaux mécanismes vous sont offerts
  - définir une DTD (Document Type Definition).
  - utiliser un schema (fichier XML de définition de grammaire)

Grammaire

Exemple XHTML avec DTD externe

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

Exemple XML avec DTD interne

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE amis [
<!ELEMENT amis (ami+)>
<!ELEMENT ami (email*,telephone?)>
   <!ATTLIST ami nom CDATA #REQUIRED
                 prenom CDATA #IMPLIED>
<!ELEMENT email (#PCDATA)>
<!ELEMENT telephone (#PCDATA)> ]>
<amis>
      <ami nom="Dupond">
      <email>dupond@example.net</email>
…
</amis>
```

Ex de DTD

```
<!ELEMENT FORMATION (TITRE, DUREE, PREREQUIS, OBJECTIF, CONTENU,
                     ANIMATEUR+)>
<!ELEMENT CONTENU (CHAPITRE)*>
<!ELEMENT CHAPITRE (TITRE, SECTION*)>

<!ELEMENT TITRE (#PCDATA)>
<!ELEMENT DUREE (#PCDATA)>
<!ELEMENT PREREQUIS (#PCDATA)>
<!ELEMENT OBJECTIF (#PCDATA)>
<!ELEMENT SECTION (#PCDATA)>
<!ELEMENT ANIMATEUR (#PCDATA)>
```

Ex de DTD

- **(#PCDATA - Parsed Character Data)**
  - ces tags ne peuvent que contenir des données textuelles
- **<!ELEMENT FORMATION (TITRE, DUREE, PREREQUIS, OBJECTIF, CONTENU, ANIMATEUR+)>**
  - le tag racine (<FORMATION>) va contenir
    - *un titre (et obligatoirement un),*
    - *puis une durée,*
    - *un pré-requis,*
    - *un objectif,*
    - *un contenu*
    - *et des animateurs (un ou plus)*
- **<!ELEMENT CONTENU (CHAPITRE)*>**
  - le contenu est lui même composé. Il peut contenir un nombre quelconque de chapitre (0 ou plus)
- **<!ELEMENT CHAPITRE (TITRE, SECTION*)>**
  - Un chapitre étant constitué d'un titre et d'autant de sections que nécessaire

Validation d'un shéma XML

- **outils de validation, pour vérifier qu'un document de données et bien conforme à la grammaire (DTD ou Schema) du langage**
  - XMLint
  - XMLSpy
  - ValidateXHTML
  - …
- **la validation de la grammaire n'est pas une étape obligatoire pour pouvoir présenter un document.**

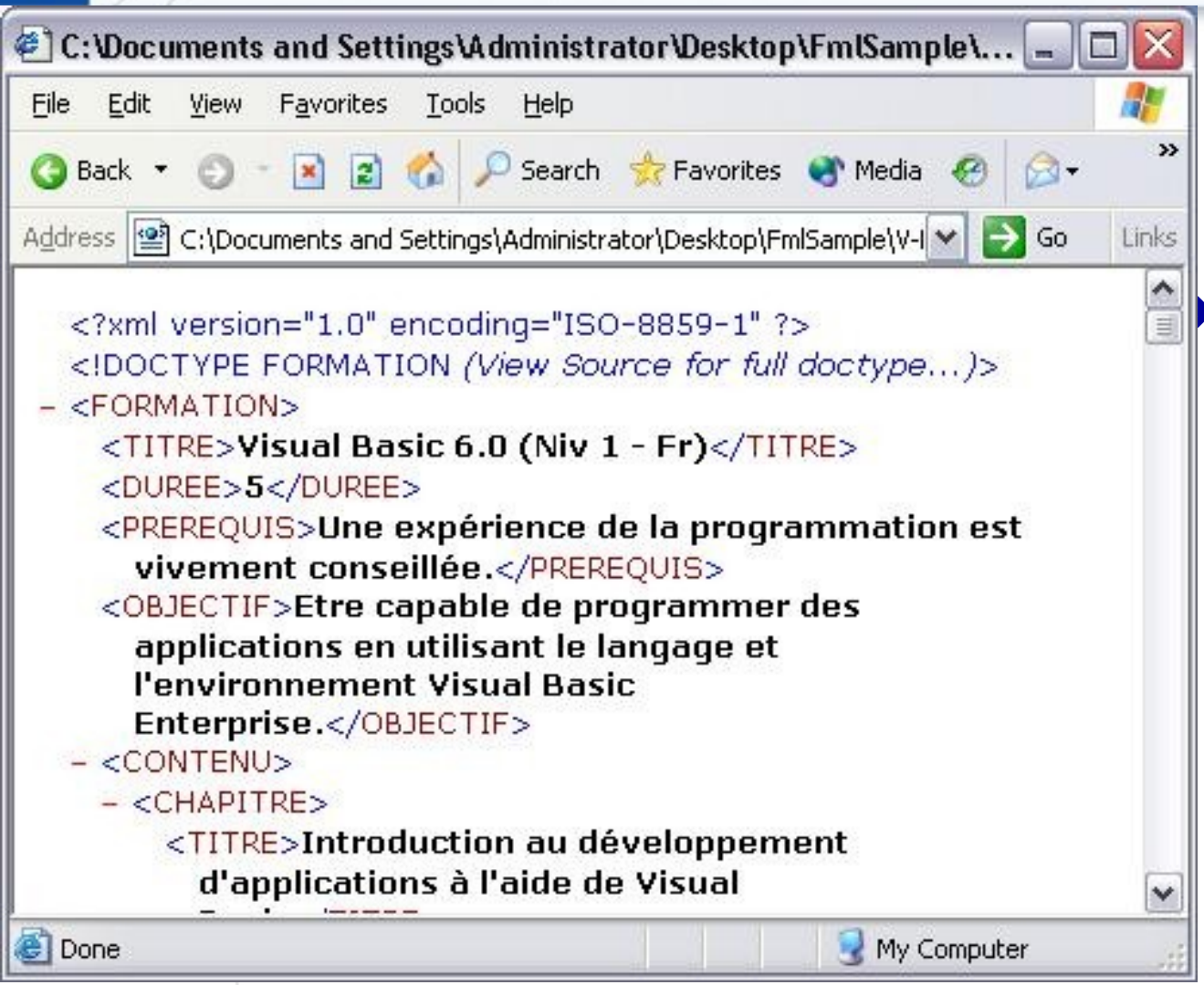
Affichage et Mise en page d'un doc XML

- les tags dont nous avons parlé dans FML on été choisis par nos soins.
  - Comment le navigateur pourrait t'il les connaître ?
- Tout comme il nous a fallut spécifier qu'elles étaient les règles d'imbrications des tags, il va nous falloir définir comment chaque tag va se présenter
- Si nous ne spécifions rien, le navigateur ne saura pas quoi faire.

Par défaut,les navigateurs se contentent simplement d'afficher le fichier XML en question.

- Alors que le Html a été conçu pour afficher de l'information, le XML a été créé pour structurer de l'information. Il ne fait rien d'autre !
- Le XML n'est que de l'information encodée entre des balises. Il faudra d'autres éléments, comme par exemple ...un fichier XSL, pour que le navigateur puisse "comprendre" vos balises et afficher ce fichier sous une forme plus conviviale

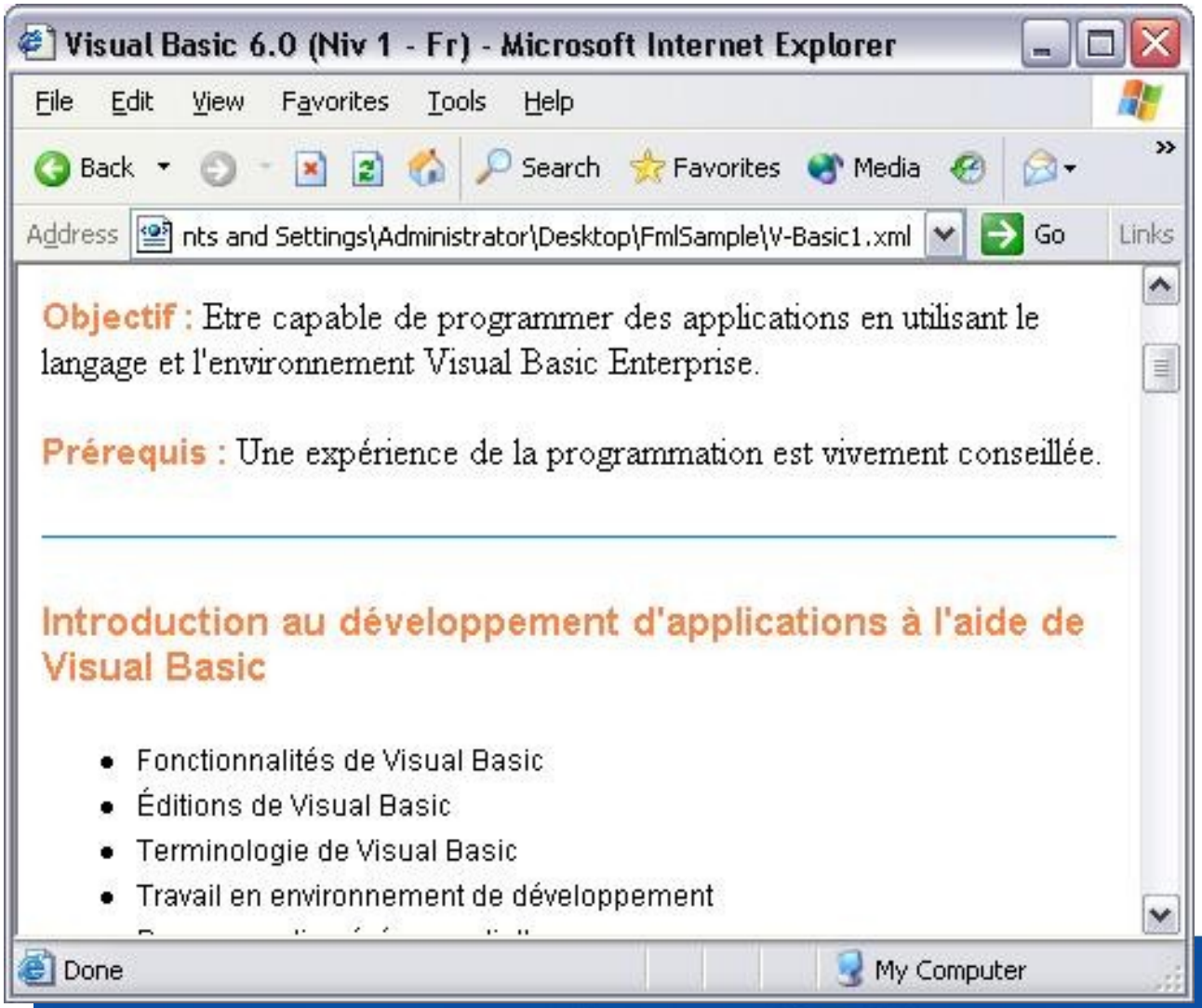
Affichage et Mise en page d'un doc XML



- *Résultat si on ne spécifie pas le tag qui précise la feuille de style*

Exemple : mise en page

- *Préciser feuille de stype*
  - *Ajouter : <?xml:**stylesheet** type="text/xsl" **href**="Formation.xsl" ?>*



Exemple : mise en page

- **Ligne d'affectation feuille de stype**
  - *<?xml:**stylesheet** type="text/xsl" **href**="Formation.xsl" ?>*

- Cette ligne d'affectation de feuille de styles contient principalement deux parties.
  - La première indique le langage de feuille de style utilisé.

- deux langages potentiellement utilisables avec XML :

- ***CSS*** (Cascading StyleSheet) et

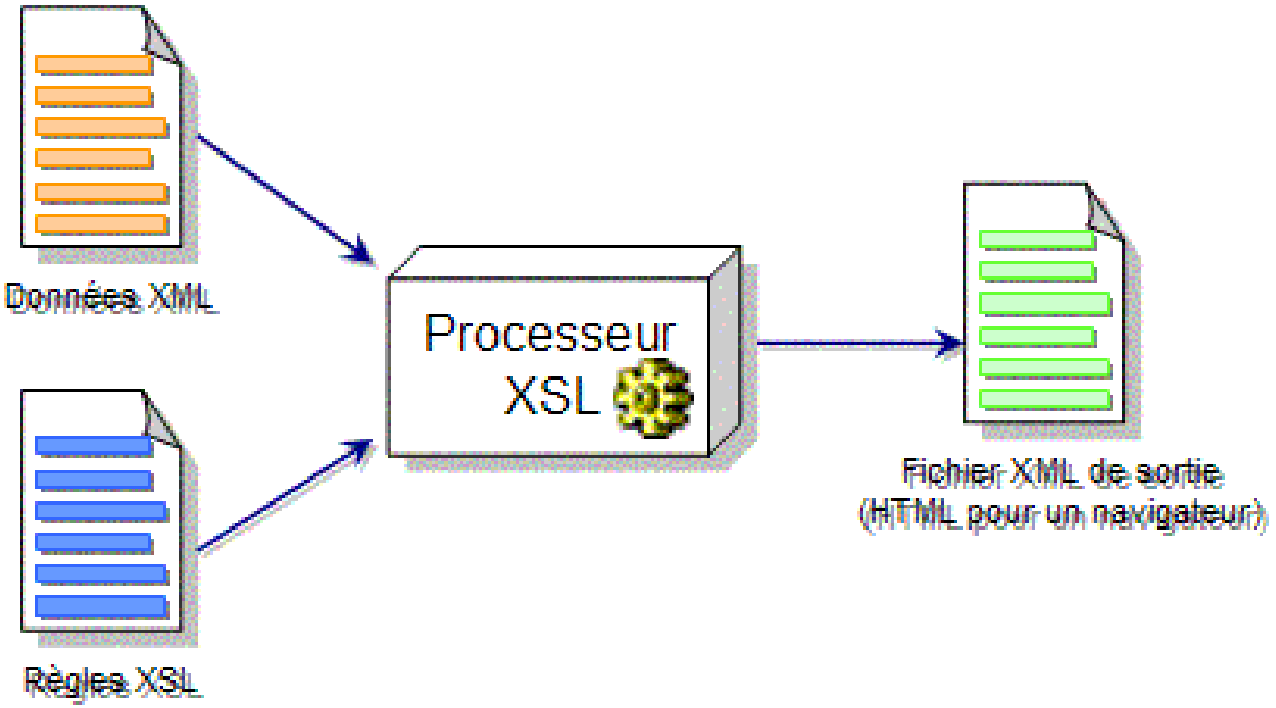
- ***XSL*** (eXtensible ***stylesheet*** Language).

  - La deuxième information à fournir est la localisation du fichier de style

- (href signifiant Hypertext REFerence).

Exemple : mise en page

- XSL prend en entrée
  - un fichier de données XML
  - un fichier de règles XSL
- après de multiples transformations, renvoie
  - un autre fichier XML.



Exemple : mise en page CSS

```
<style type="text/css">
formation {}
titre {
display: block;
width: 250px;
font-size: 16pt ;
font-family: arial ;
font-weight: bold;
background-color: teal;
color: white;
padding-left: 10px;
}
duree {
display: block;
font-size: 12pt;
padding-left: 10px;
}
…
</style>
```

Exemple : mise en page XSL

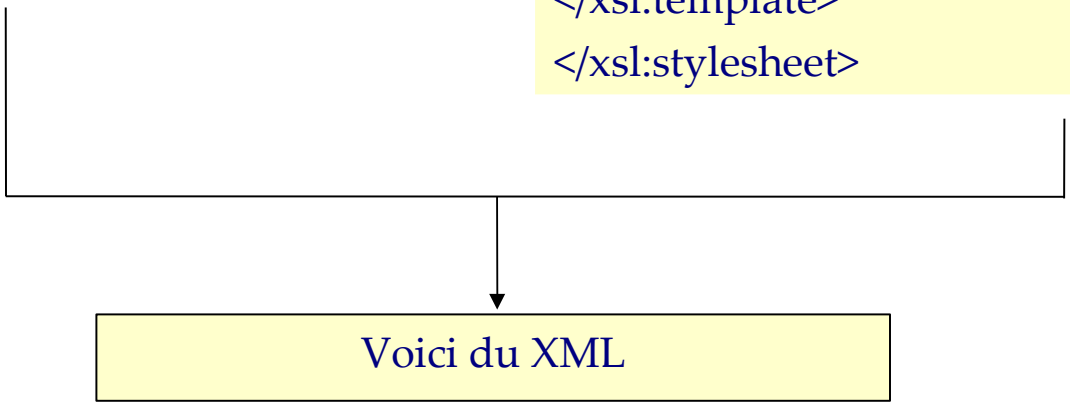
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output encoding="ISO-8859-1"/>
<!-- Toutes les autres règles de la feuille de styles -->
  <xsl:template match="CHAPITRE">
      <H2><xsl:value-of select="./TITRE"/></H2>
      <UL> <xsl:apply-templates select="./SECTION"/> </UL>
  </xsl:template>
  <xsl:template match="SECTION">
      <LI><xsl:value-of select="."/></LI>
    </xsl:template>
</xsl:stylesheet>
```

XSL (*eXtensible Stylesheet Language*)

```
<?xml version="1.0"?>
<?xml-stylesheet href="fichierxsl.xsl"?>
<demoXML>
<message>Voici du XML</message>
</demoXML>
```

+

```
<?xml version="1.0"?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
<xsl:value-of select="demoXML/message"/>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```



Bcp de travail pour peu de résultats :-(… mais

XSL : principe

- *Le XSL ne permet pas uniquement l'affichage de XML. Il permet aussi :*
  - de sélectionner une partie des éléments XML
  - de trier des éléments XML
  - de filtrer des éléments XML en fonction de certains critères
  - de choisir des éléments
  - de retenir des éléments par des tests conditionnels
- **Principe de fonctionnement**
- le XSL est dérivé du XML.
  - doc XSL reprend structure & syntaxe de n'importe quel doc XML
- le document XSL comporte un document Xhtml qui sera quant à lui reconnu par le navigateur et qui servira de support à tout ou partie des données du document XML associé
- le XSL fonctionne avec une ou plusieurs "templates", sorte de gabarit pour définir comment afficher des éléments du fichier XML. Les éléments concernés du fichier XML sont déterminés par l'attribut "match".

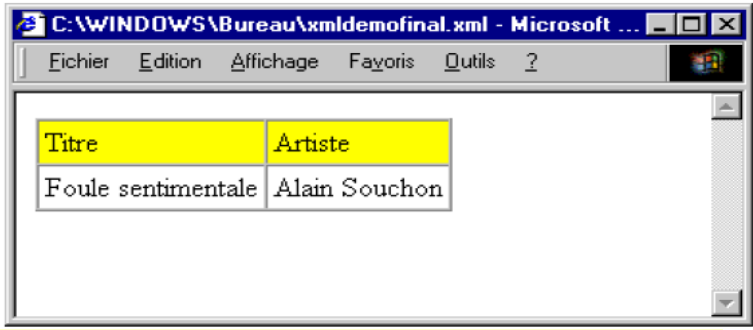
XSL : un exemple

- <?xml version="1.0"?>
  - XSL est dérivé du XML ==> doc XSL commence par déclaration XML
- <xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
- le doc est du XSL extensible stylesheet
  - L'attribut xmlns fait référence au namespace" utilis
  - namespace du W3C: xmlns:xsl="http://www.w3.org/1999/XSL/Transform
- <xsl:template match="/">
  - balise template et son attribut match
  - Cette balise template va déterminer un gabarit dans lequel on va transformer des éléments du fichier XML sous une forme que le navigateur pourra afficher
  - Les éléments du fichier XML sont déterminés par l'attribut match="/".
  - Le "/" signale que sont concernées toutes les balises XML du document associé à partir de la racine

XSL : un exemple

- <html> <body>
  - Début partie Html qui servira de support pour l'affichage du document dans le navigateur
  - Ensuite .... diverses balises Html et XSL... Par exemple :
- <xsl:value-of select="chemin d'accès/élément"/>
  - <xsl:value-of> : permet de sélectionner un élément du fichier XML associé pour le traiter dans le fichier XSL.
  - Dans l'attribut **select**, on détermine chemin d'accès vers la balise XML souhaitée (puisque le XML est structuré)
- </body> </html> //Fin de la partie en Html.
- </xsl:template>
  - fermeture de la balise de template.
- </xsl:stylesheet>
  - Le document XSL se termine obligatoirement par la fermeture de la balise de déclaration de document XSL

XSL : un autre exemple

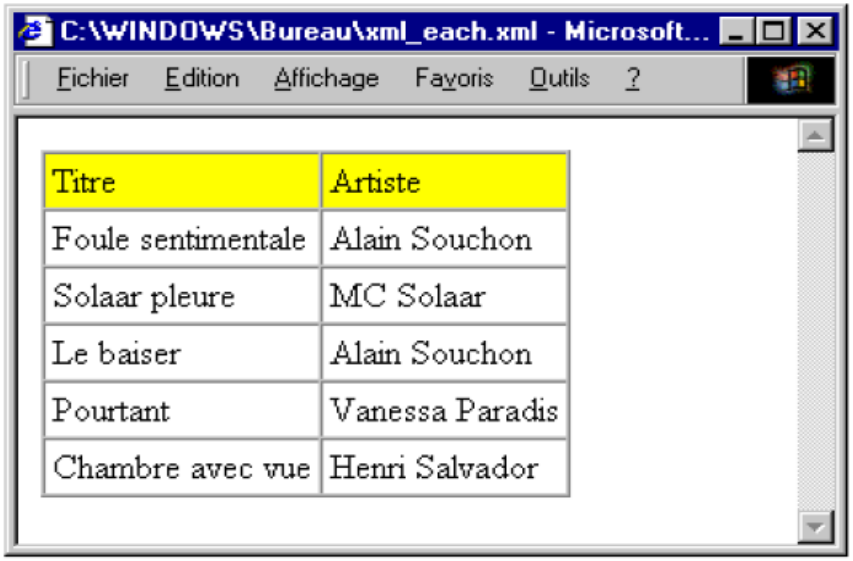



```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
    href="xsldemo.xsl"?>
<compilation>
<mp3>
<titre>Foule sentimentale</titre>
<artiste>Alain Souchon</artiste>
</mp3>
<mp3>
<titre>Solaar pleure</titre>
<artiste>MC Solaar</artiste>
</mp3>
<mp3>
<titre>Chambre avec vue</titre>
<artiste>Henri Salvador</artiste>
</mp3>
</compilation>
```

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
<table border="1" cellspacing="0" cellpadding="3">
<tr bgcolor="#FFFF00">
<td>Titre</td>
<td>Artiste</td>
</tr>
<tr>
<td><xsl:value-of select="compilation/mp3/titre"/></td>
<td><xsl:value-of select="compilation/mp3/artiste"/></td>
</tr>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

XSL : un autre exemple (suite)

```
<?xml version='1.0'?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<body>
<table border="1" cellspacing="0" cellpadding="3">
<tr bgcolor="#FFFF00">
<td>Titre</td>
<td>Artiste</td>
</tr>
<xsl:for-each select="compilation/mp3">
<tr>
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="artiste"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

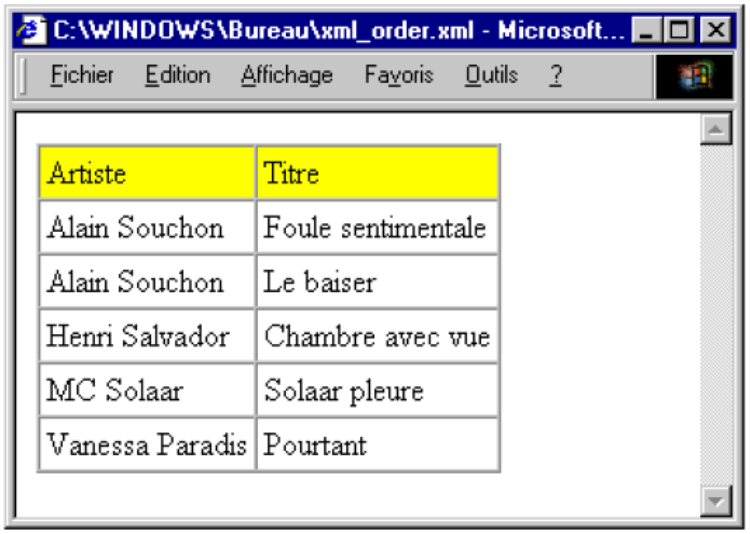


Titre	Artiste
Foule sentimentale	Alain Souchon
Solaar pleure	MC Solaar
Le baiser	Alain Souchon
Pourtant	Vanessa Paradis
Chambre avec vue	Henri Salvador

Trier avec XSL

- **order-by="+balise"** : trier par ordre croissant (+) les données comprises entre la balise désignée"
- **order-by="-balise"** : trier en ordre décroissant

  - <xsl:for-each select="compilation/mp3" **order-by="+artiste"**>

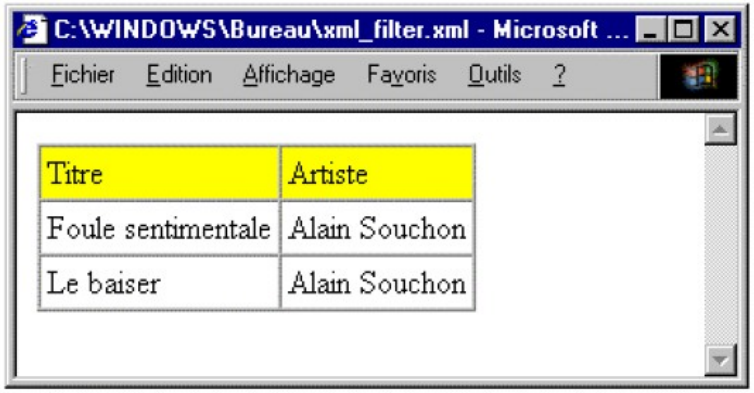


Artiste	Titre
Alain Souchon	Foule sentimentale
Alain Souchon	Le baiser
Henri Salvador	Chambre avec vue
MC Solaar	Solaar pleure
Vanessa Paradis	Pourtant

Filtrer avec XSL

- XSL permet de filtrer les données du fichier XML associé selon des critères comme égal (=), pas égal (!=), plus grand que (&gt), plus petit que (&lt)
- Utiliser l'attribut select="chemin_d'accès[balise='xxx']"

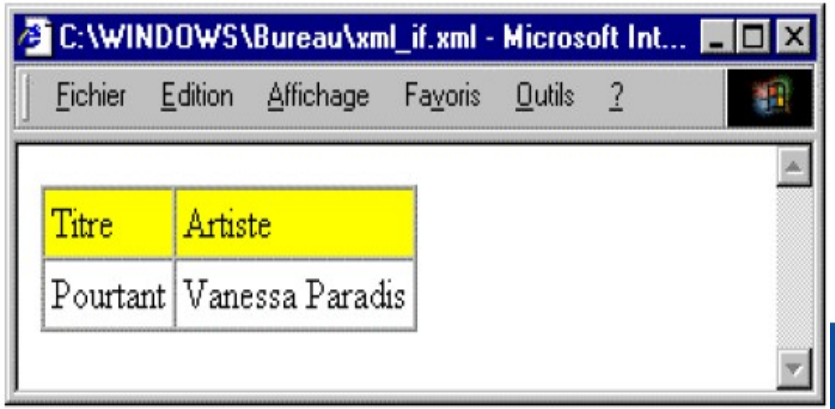
```
<xsl:for-each select="compilation/mp3[artiste='Alain Souchon']">
```



Choisir avec XSL

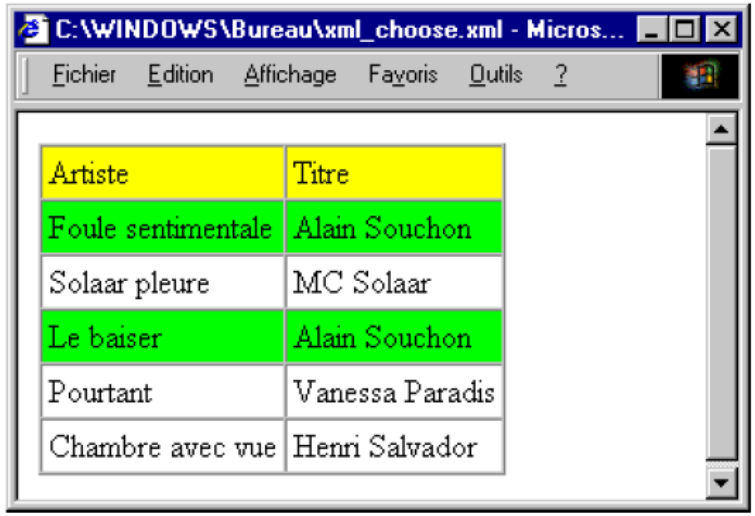
- **<xsl:if match=".[balise='xxx']">** balises Html **</xsl:if>**
- effectuer un choix dans les données du fichier XML
- On ajoutera l'attribut match où l'on indique l'élément choisi

```
<xsl:for-each select="compilation/mp3">
<xsl:if match=".[artiste='Vanessa Paradis']">
<tr>
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="artiste"/></td>
</tr>
</xsl:if>
</xsl:for-each>
```



Choix conditionnels avec XSL

```
<xsl:choose>
 <xsl:when test=".[artiste='Alain Souchon']">
<tr bgcolor="#00FF00">
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="artiste"/></td>
</tr>
</xsl:when>
<xsl:otherwise>
<tr>
<td><xsl:value-of select="titre"/></td>
<td><xsl:value-of select="artiste"/></td>
</tr>
</xsl:otherwise>
</xsl:choose>
```



Javascript et XSL

- Possibilité d'utiliser un script en Javascript pour faire la transformation du XML & XSL en Html
  - Construire fichier Html avec code Javascript qui va transformer nos fichiers XML et XSL en un seul fichier Html

```
<body>
<script type="text/javascript">
// chargement du fichier XML
var xml = new ActiveXObject("Microsoft.XMLDOM")
xml.async = false
xml.load("xmldemo.xml")
// chargement du fichier XSL
var xsl = new ActiveXObject("Microsoft.XMLDOM")
xsl.async = false
xsl.load("xsl_choose.xsl")
// transformation en Html
document.write(xml.transformNode(xsl))
</script>
</body> </html>
```

XSL : langages

- Le XSL comporte 3 langages :
  - **XSLT**
    - *langage qui **T**ransforme un document XML en un format, généralement en Html, reconnu par un navigateur*
  - **Xpath**
    - *permet de définir & adresser des parties de document XML*
  - **XML Formatter**
    - *pour "formater" du XML (transformé) de façon qu'il puisse être rendu sur des PCpockets ou des unités de reconnaissance vocale*

Xpath : principe

- Langage d'interrogation pour localiser une portion d'un document XML
- Une expression XPath est un chemin constitué de *pas de localisation*.
- Les pas de localisation ont chacun trois composants :
  - 1. **un axe** ;
    - *indique la direction dans laquelle se déplacer dans l'arbre XML, relativement au nœud courant ou depuis la racine.*
    - *ex,* child:: *sélectionne nœuds enfants du nœud courant*
    - *Un autre axe largement utilisé est celui des attributs, représenté avec le caractère @*
  - 2. **un test de nœud** ;
    - *sélectionner nœuds en fonction de leur nom ou type.*
    - *ex,* text() *sélectionne les nœuds de type texte*
  - 3. **des prédicats**
    - *expressions utilisés pour filtrer les nœuds sélectionnés par l'axe et le test de nœud.*
    - *Les prédicats sont écrits entre crochets (« [», «] »). Si le prédicat est évalué à vrai, les nœuds correspondants seront sélectionnés.*

Xpath : exemple

```
<?xml version="1.0"?>
 <racine>
  <cours nom="rsx102" site="http://cnam.fr/">
   <article nom="XPath"></article>
  </cours>
 </racine>
```

/	sélectionne un nœud "fictif", qui englobe tout le document
//article	sélectionne tous les éléments "article" du document où qu'ils soient
/racine/cours	sélectionne l'unique élément "cours" puisqu'il est ici le seul fils de "racine" portant ce nom
//article[@nom='XPath']	sélectionne tous les éléments "article" du document où qu'ils soient, ayant un attribut "nom" dont la valeur est "XPath"

Le résultat de ces sélections dépendra de la nature de la tâche :

* En **affichage**: ce sera la valeur textuelle, propre à chaque type d'élément, qui apparaîtra
* En **sélection**, il se comportera comme un pointeur sur lequel d'autres requêtes XPath pourront être effectuées.

Xquery : principe

- Langage de requête permettant (W3C)
  - d'extraire des infos d'un doc ou d'une collection de docs XML
  - d'effectuer des calculs complexes à partir des informations extraites et de reconstruire de nouveaux documents ou fragments XML.

- XQuery joue par rapport aux données XML un rôle similaire à celui du langage SQL vis-à-vis des données relationnelles ...

- Syntaxes
  - FLWOR : … for, let, where, order by et return
  - XQueryX (XML Syntax for Xquery) dans laquelle
    - une requête est un document XML.
    - De ce fait, elle est beaucoup plus verbeuse et moins lisible que la précédente
    - est destinée à des manipulations formelles par des programmes

Xquery : un exemple

```
<employes>
  <employe>
    <nom>Dupond</nom>
    <prenom>Albert</prenom>
    <date_naissance>23/09/1958</date_naissance>
  </employe>
  <employe>
    <nom>Dupont</nom>
    <prenom>Alphonse</prenom>
    <date_naissance>23/12/1975</date_naissance>
  </employe>
  <employe>
    <nom>Dupont</nom>
    <prenom>Isabelle</prenom>
    <date_naissance>12/03/1967</date_naissance>
  </employe>
...
</employes>
```

```
for $b in document
("http://example.com/exemple.xml")//employe
where $b/nom = "Dupont"
return
  <dupont>{
    $b/prenom,
    $b/date_naissance
  }</dupont>
```

```
<dupont>
  <prenom>Alphonse</prenom>
  <date_naissance>23/12/1975</date_naissance>
</dupont>
<dupont>
  <prenom>Isabelle</prenom>
  <date_naissance>12/03/1967</date_naissance>
</dupont>
```

Exemple : C/C

➔ XML,

- permet de définir langage de balisage ***adapté*** à vos besoins

➔ Pour ce faire, il vous faut définir

- une ***grammaire*** ➔ comment correctement utiliser les tags

- Des outils de validations de grammaire sont dispo.

- Deux sous langages pour définir grammaire :

- une DTD (Document Type Definition) ou

- un Schéma (grammaire XML bien définie)

➔ Possibilité d'adjoindre une ***feuille de styles*** à votre langage,

- cela vous permettant de présenter vos données.
- deux langages sont utilisables :

- CSS (Cascading StyleSheet) et

- XSL (eXtensible Stylesheet Language). (plus puissant que CSS)

Plan

- Introduction au langage XML
- Un premier exemple
- Règle de syntaxe XML
- Mise en œuvre d'une DTD

Syntaxe : règles de base

- un fichier XML utilise des tags pour structurer ses données.
  - Différentes règles existent pour régire l'utilisation des tags

- **La notion de tags XML**
  - A l'instar d'HTML, XML utilise aussi des tags (marques/balises).
  - Un tag commence par le caractère < et se termine par le caractère >.
    - tags ouvrant : ex <CHAPITRE>
    - tags fermant : ex </CHAPITRE>

- **règles syntaxiques élémentaires**
  - *Règle n° 1* : si le fichier est syntaxiquement incorrect, le traitement du fichier aboutira à une erreur.
  - *Règle n° 2* : Tout tag ouvrant *doit obligatoirement* être fermé plus loin dans le fichier (≠ HTML)

Syntaxe : règles de base

- forme compacte pour représenter un couple de tags XML ne contenant pas de données.
  - Il suffit du fournir un unique tag commençant par le caractère < et se terminant par les caractères />

<DATE Jour="26" Mois="08" Année="1973"**></DATE>**
<DATE Jour="26" Mois="08" Année="1973" **/>**

  - *Règle n° 3* : distinction entre lettres minuscules et lettres majuscule
  - *Règle n° 4* : nom de tags

    - ne doit pas commencer par XML

    - ne contient que lettres, chiffres, -, _, . et :

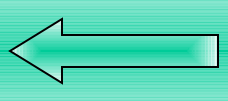
    - ne peuvent commencer que par un _ ou une lettre

Syntaxe : imbrication des tags XML

- *Règle n° 5* : les tags ne devrons en aucun cas se chevaucher

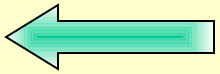
  - ils se doivent d'être imbriqués les uns dans les autres.

```
<TAG-PARENT>
    <TAG-ENFANT>
</TAG-PARENT>
    </TAG-ENFANT>
```



FAUX

```
<DATE>
    <JOUR>26</JOUR>
    <MOIS>08</MOIS>
    <ANNEE>1973</ANNEE>
</DATE>
```

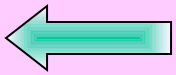


OK

- *Règle n° 6* : tout doc. XML, doit et ne peut avoir qu'un seul tag racine

  - Tous les autres tags du doc. se devront d'être contenus dans ce tag

```
<?xml version="1.0" ?>
<TAG-RACINE> <!-- Contenu du tag --> </TAG-RACINE>
<AUTRE-TAG-RACINE> <!-- Contenu du tag --> </AUTRE-TAG-RACINE>
```



FAUX

Syntaxe : bonnes pratiques

- opter pour nom de tags clairs et explicites (même s'ils sont plus long)
- bien comprendre ce que vous devez faire des données comprises dans vos tags.
  - Le choix de votre structure peut influer sur la suite des choses (comment récupérer les données dans une feuille de styles, par ex).

```
<DATE>26/08/1973</DATE>
            ou
<DATE>
            <JOUR>26</JOUR>
            <MOIS>08</MOIS>
            <ANNEE>1973</ANNEE>
 </DATE>
```

Syntaxe : les attributs

- Les attributs d'un tag sont fournis à la suite du nom du tag
  - séparés par un espace.
- attribut = deux parties : nom et valeur.
  - La valeur doit être comprise soit entre des simples **quottes** soit entre des doubles **guillemets**.
    - *<date anniversaire=071185> ==> incorrect*
    - *<date anniversaire="071185"> ==> correct*
  - Le nom est séparé de la valeur par le signe d'égalité.

```
<TagName attribut1="valeur1" attribut2='valeur2'>
    Donnée du tag
</TagName>
```

Syntaxe : les attributs

- valeur d'attributs ou données de sous-tags ?
  - un attribut vient ***qualifier*** la données d'un tag. Cela rajoute un ***complément d'information*** utile pour le traitement de la données.
  - un attribut ***ne doit pas contenir la donnée principale***.
  - Ex : dans le cas d'une date, il est clair que l'information principale est constituée du jour, du mois et de l'année.
    - *Ces trois infos devraient donc normalement être des données de tags*

```
<DATE>
      <JOUR>26</JOUR>
      <MOIS>08</MOIS>
      <ANNEE>1973</ANNEE>
 </DATE>
```

est donc préférable à

```
<DATE>26/08/1973</DATE>
```

ou alors (pour feuille de style)

```
<DATE format="fr">26/08/1973</DATE>
```

Syntaxe : attributs prédéfinis

- **xml:lang** ➔ langue utilisée dans une partie du document
- **xml:space** ➔ caractères de séparation.

➔ obligation de les redéfinir dans la DTD si validation (sémantique)

➔ attribut et valeur sont cascadés sur tous les sous-tags.

- Il est donc possible de ne spécifier qu'une unique fois xml:lang (par ex) pour tout le document : il suffit de le définir sur le tag racine.

  - <EXEMPLE xml:lang="**fr**">Aquarelle</EXEMPLE>

Syntaxe : Version du XML et système d'encodage

- **<?xml version="1.0" ?>** ➔ 1ère ligne : version 1.0, 1.1, 1.2 ou 1.3
  - Indique aux outils XML norme codage (ISO-10646), ….

- **xml:space** ➔ caractères de séparation.

➔ obligation de les redéfinir dans la DTD si validation (sémantique)

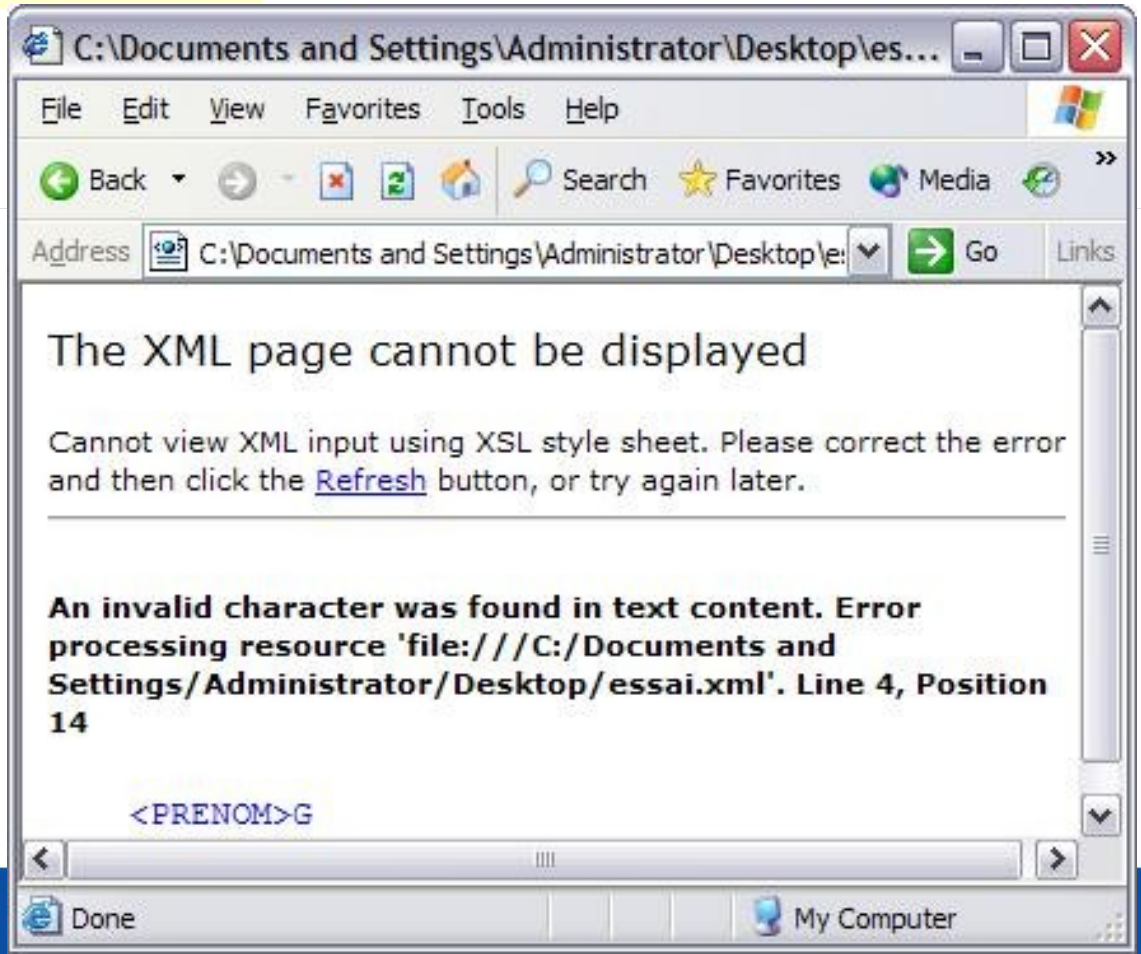
➔ attribut et valeur sont cascadés sur tous les sous-tags.

- Il est donc possible de ne spécifier qu'une unique fois xml:lang (par ex) pour tout le document ➔ il suffit de le définir sur le tag racine.

➔ Ne rien spécifier ➔ norme ASCII ➔ Si accents Alors message d'erreur

Syntaxe : Version du XML et système d'encodage

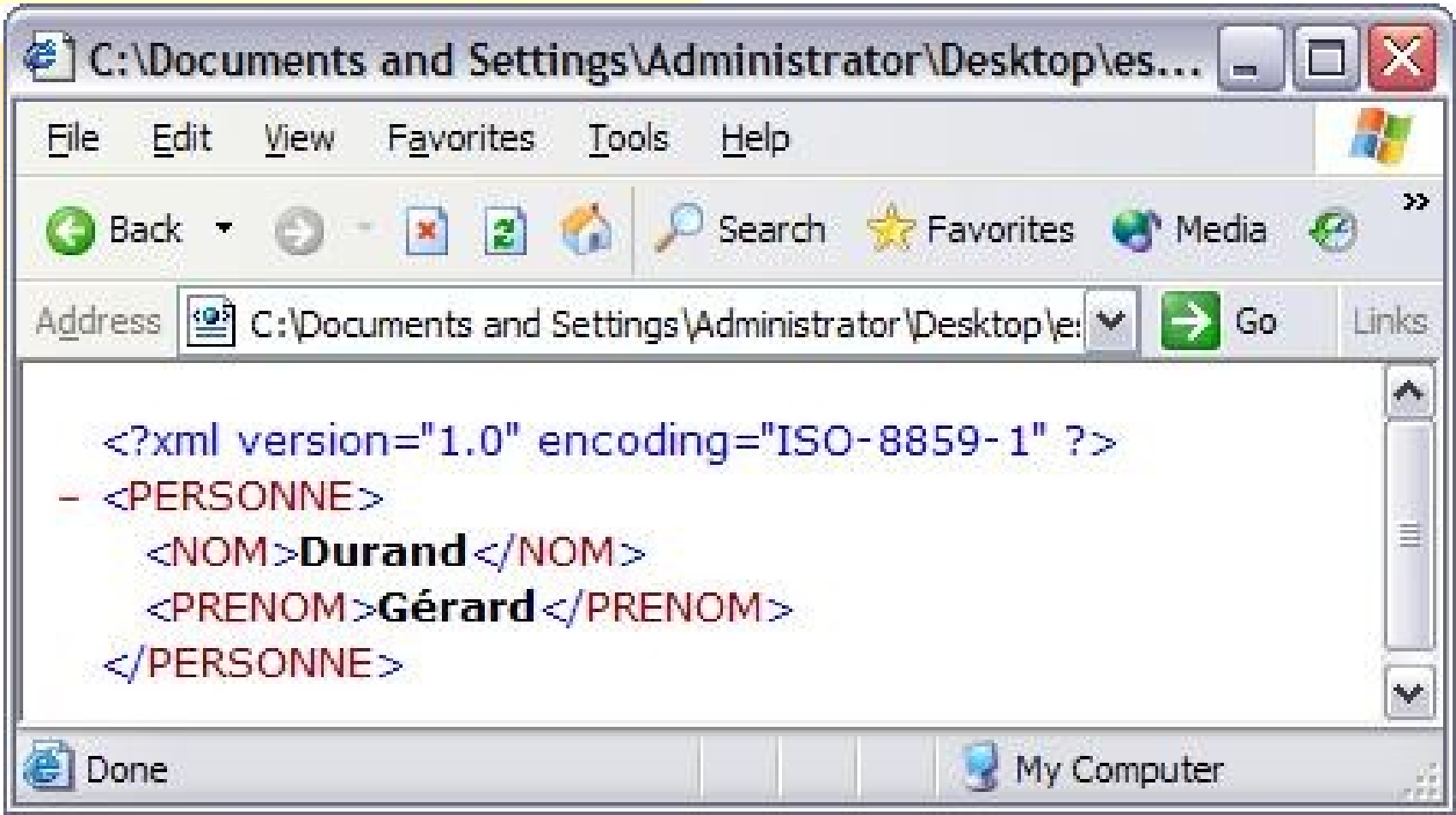
```
<?xml version="1.0" ?>
<PERSONNE>
      <NOM>Durand</NOM>
      <PRENOM>Gérard</PRENOM>
</PERSONNE>
```



Syntaxe : Version du XML et système d'encodage

```
<?xml version="1.0" encoding="ISO-8859-1" ?> <PERSONNE>
    <NOM>Durand</NOM>
    <PRENOM>Gérard</PRENOM>
</PERSONNE>
```

spécifier le système d'encodage ayant servit à générer le fichier



Syntaxe : Liaison à une feuille de styles

- **Feuille de style** ➔ Affichage dans navigateur
  - *langages CSS ou XSL*
- Liaison ➔ rajouter "<?xml:stylesheet" ….
- L'attribut ***type*** : spécifier *langage* utilisé ("text/css" ou "text/xsl")
- L'attribut ***href*** indique la *localisation* du fichier

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

<?xml:stylesheet type="text/css" href="Personnes.css" ?>

<PERSONNE>
    <NOM>Durand</NOM>    <PRENOM>Gérard</PRENOM>
</PERSONNE
```

Syntaxe : Liaison à une DTD

- **DTD** ➔ définir une grammaire XML
- Outils de validation ➔ contrôler la bonne utilisation des tags dans un fichier
- DTD peut directement être embarquée dans le fichier de données.
  - Dans ce cas, la DTD ne sert que pour cet unique fichier
- *nom* associé à la *DTD* doit être exactement **identique** au *nom* du *tag racine*,

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

<!DOCTYPE PERSONNE SYSTEM "Personne.dtd" [] >

<PERSONNE>
      <NOM>toto</NOM>
     <PRENOM>titi</PRENOM>
</PERSONNE
```

Syntaxe : notion de namespaces

- Deux grammaires XML distinctes définissent des tags qui pourraient vous convenir
  - *Est-il possible de définir une grammaire qui inclue ces deux langages* ?
- Considérons aussi, que chacune des deux grammaires sur lesquelles vous vous appuyez définissent un même nom de tag, par ex <DATE>, mais dont leurs utilisations et leurs significations seront différentes d'un langage à un autre.
  - *Comment dire que l'on cherche à utiliser l'un des deux tags ?*

notion d'espaces de noms

- nom complet, de tag est constitué de deux parties séparées par "**:**".
  - 1ère partie spécifie dans quel espace de noms le tag est défini
  - 2nd nomme le tag.
  - *On peut avoir* <**FirstLang:DATE**> *et* <**SecondLang:DATE**> *sans risque de confusion.*

Syntaxe : notion de namespaces

- Pour garantir l'unicité du tag, un espace de noms est généralement associé à une URL
  - chaque société possède bien sa propre URL.
    - *deux sociétés peuvent définir deux langages utilisant des tags similaires sans risque de conflits.*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        version="1.0">
    <xsl:output encoding="ISO-8859-1"/>
    <xsl:template match="/">
    <xsl:apply-templates/>
    </xsl:template>
    <!-- Suite -->
</xsl:stylesheet>
```

Syntaxe : Injection de caractères Unicode

- Il est possible d'insérer n'importe quel caractère Unicode.
  - "&#" valeur Unicode du caractère souhaité puis ";".

```
<CONTENT>
Quelques caractères Grecque : &#931; &#928; &#934;
&#937;
</CONTENT>
```

Quelques caractères Grecque : Σ Π Φ Ω

Syntaxe : Utilisation des entités prédéfinies

- pour structurer flux XML, il vous faut, au moins, insérer des tags.
  - *dans ce cas, comment insérer caractère d'infériorité dans vos données ?*

<	Less than	<
>	Greater than	>
&	Ampersand	&
"	Quote	"
'	Apostrophe	'

Plan

- Introduction au langage XML
- Un premier exemple
- Règle de syntaxe XML
- Mise en œuvre d'une DTD

Mise en oeuvre d'une DTD : intro

- même si syntaxe correcte, ce n'est pas pour autant que vous les avez utilisé correctement les un par rapport aux autres, relativement à leur sémantique !!

- Pour **valider** que vos fichiers de données sont **grammaticalement correcte**, vous allez utilisez un langage de définition de grammaire XML.
  - DTD
  - langage de schémas

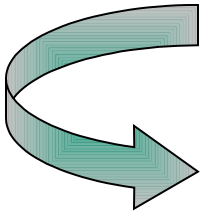
Mise en oeuvre d'une DTD : définitions

document bien formé

- respecte bien toutes les règles de syntaxes XML.

document valide

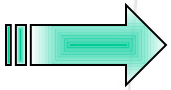
- doc bien formé + respect une grammaire donnée



Voir environnement de travail XMLSpy

Ou définir une DTD ?

- Il existe deux possibilités pour définir une DTD.
  - Soit vous embarquez votre DTD au sein d'un fichier de données,
  - soit la définir dans un fichier externe.
  - Les deux techniques peuvent même être mixées.



- La 2nde alternative est préférable.
  - plus simple de partager votre DTD pour un ensemble de documents.

- dans les deux cas, que votre DTD va devoir être nommée.
  - Ce nom se doit obligatoirement d'être celui du tag racine de vos fichiers de données (aux minuscules et majuscules près).

Définition d'une DTD embarquée

- Si vous embarquez votre DTD dans un fichier de données, elle sera alors localisée dans le prologue, au niveau du tag *<! DOCTYPE ...>*.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE exemple [
        <!-- début de la DTD --> . . .
        <!-- fin de la DTD -->
]>
<exemple>
        <!-- Suite du document XML -->
</exemple>
```

Définition d'une DTD dans fichier externe

- Pour lier un fichier de données à une DTD, il vous faut aussi utiliser le tag **<!DOCTYPE ... >**
- mot clé **SYSTEM** utilisé pour indiquer ou se trouve la DTD
  - c'est une URL qui est attendue.
  - Vous pouvez donc lier vos fichiers à une quelconque DTD présente sur le Web.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE exemple SYSTEM "fichier.dtd" []>
<exemple>
        <!-- Suite du document XML -->
</exemple>
```

Définition des règles d'utilisation des tags

- une DTD définie
  - des tags et leurs règles d'imbrication,
  - des listes d'attributs et des entités
  - …
- Exemple : petit langage de définition de tableaux

```
<TABLEAU>
    <TITRE>Titre du tableau</TITRE>
    <LIGNE>
        <CELL-E>\</CELL-E>
        <CELL-E>Statistique 1</CELL-E>
        <CELL-E>Statistique 2</CELL-E>
    </LIGNE>
    <LIGNE>
        <CELL-E>Expérience 1</CELL-E>
        <CELL-D>25</CELL-D>
        <CELL-D>34</CELL-D>
    </LIGNE>
</TABLEAU>
```

- **<CELL-E>** : cellules d'entête (titre).
- **<CELL-D>** : cellules de données

Définition de tags contenant des données

- deux types de tags :
  - ceux qui contiennent des données textuelles
  - ceux qui contiennent des sous tags

Cas 1 : tags qui contiennent des données textuelles

```
TABLEAU>
    <TITRE>Titre du tableau</TITRE>
    <LIGNE>
        <CELL-E>\</CELL-E>
        <CELL-E>Statistique 1</CELL-E>
        <CELL-E>Statistique 2</CELL-E>
    </LIGNE>
    <LIGNE>
        <CELL-E>Expérience 1</CELL-E>
        <CELL-D>25</CELL-D>
        <CELL-D>34</CELL-D>
    </LIGNE>
</TABLEAU>
```

- **ELEMENT** : on cherche à définir un tag

```
<!ELEMENT TITRE (#PCDATA)>
<!ELEMENT CELL-E (#PCDATA)>
<!ELEMENT CELL-D (#PCDATA)>
```

- **#PCDATA** : tag contiendra des données *'Parsed Character DATA'*

Définition de tags contenant des sous tags

?	*Occurrence* : peut apparaître 0 ou 1 fois.
+	*Occurrence* : peut apparaître 1 ou plusieurs fois.
*	*Occurrence* : peut apparaître 0 ou plusieurs fois.
,	*Séquence* : Les 2 parties doivent apparaître et dans cet ordre
|	*Choix* : permet de choisir entre deux alternatives.
ANY	*Choix* : le tag peut contenir n'importe quoi.
EMPTY	Le tag ne peut strictement rien contenir.

Exemples

<!ELEMENT agenda (personne*) >	Un agenda est constitué d'un nombre indéterminé de personnes
<!ELEMENT personne (nom, prenom?, date)>	Une personne doit posséder un nom puis un prénom (facultatif) puis une date de naissance
<!ELEMENT hr (EMPTY) >	En HTML, le tag HR (Horizotal Rule) est définie comme étant unitaire : il ne contient rien.

Définition de tags contenant des données

- <!ELEMENT LIGNE (CELL-E* | CELL-D*) >
  - soit contenir nbre qlq de cellules de données soit nbre qlq de cellules d'entête
- <!ELEMENT LIGNE (CELL-E | CELL-D)* >
  - on peut avoir un nombre quelconque de cellules. Chacune de ces cellules pouvant être soit une cellule de titre, soit une cellule de données.

```
TABLEAU>
    <TITRE>Titre du tableau</TITRE>
    <LIGNE>
        <CELL-E>\</CELL-E>
        <CELL-E>Statistique 1</CELL-E>
        <CELL-E>Statistique 2</CELL-E>
    </LIGNE>
    <LIGNE>
        <CELL-E>Expérience 1</CELL-E>
        <CELL-D>25</CELL-D>
        <CELL-D>34</CELL-D>
    </LIGNE>
</TABLEAU>
```

si l'on veut fonctionner comme en HTML, la règle est :
<!ELEMENT LIGNE (CELL-E | CELL-D)* >

DTD : Définition de tags contenant des données

- un tableau peut contenir un nombre quelconque de ligne, mais au plus un seul titre (ce dernier étant facultatif).

```
TABLEAU>
        <TITRE>Titre du tableau</TITRE>
        <LIGNE>
                <CELL-E>\</CELL-E>
                <CELL-E>Statistique 1</CELL-E>
                <CELL-E>Statistique 2</CELL-E>
        </LIGNE>
        <LIGNE>
                …
        </LIGNE>
</TABLEAU>
```

```
<!ELEMENT TABLEAU    ((          TITRE?,  LIGNE*)      |
                       (LIGNE*,  TITRE?,  LIGNE*)      |
                       (LIGNE*,  TITRE?          ) )  >
```

DTD : Définition de tags contenant des données

```
TABLEAU>
    <TITRE>Titre du tableau</TITRE>
    <LIGNE>
        <CELL-E>\</CELL-E>
        <CELL-E>Statistique 1</CELL-E>
        <CELL-E>Statistique 2</CELL-E>
    </LIGNE>
    <LIGNE>
        …
    </LIGNE>
</TABLEAU>
```

```
<!ELEMENT TABLEAU     (LIGNE*,   TITRE?,  LIGNE*) >
<!ELEMENT LIGNE (CELL-E | CELL-D)* >
<!ELEMENT TITRE (#PCDATA)>
<!ELEMENT CELL-E (#PCDATA)>
<!ELEMENT CELL-D (#PCDATA)>
```

DTD : Définition de listes d'attributs

- ***ATTLIST*** ➔ introduire une liste d'attributs,
  - *ATTLIST*, suivi du *nom du tag* et des *données descriptives* de chaque attribut.

```
<!ELEMENT TABLEAU (LIGNE*, TITRE?, LIGNE*) >
<!ATTLIST   TABLEAU
      largeur CDATA "80%"
      bordure CDATA "1px"
>
<!ELEMENT LIGNE (CELL-E | CELL-D)* >
<!ELEMENT TITRE (#PCDATA)>
<!ATTLIST   TITRE
      alignement (top | bottom) "bottom"
>
<!ELEMENT CELL-E (#PCDATA)>
<!ELEMENT CELL-D (#PCDATA)>
```

attribut *largeur* :
- chaîne de caractères
- Valeurs par défaut = 80%
  - *au cas ou aucune valeur ne serait définie dans le fichier de données*

DTD : Aspects avancés de définition d'attributs

- Si vous ne fournissez pas val par défaut, trois alternatives sont offertes
  - **#REQUIRED** : valeur de l'attribut est à fournir obligatoirement
    - *Si un fichier de données contient un tag, pour lequel un attribut est **obligatoire**, et si aucune valeur n'est fournie alors le document générera un message d'erreur.*
  - **#FIXED :** forcer un attribut à une valeur donnée.
    - *l'attribut existe. Sa valeur est **imposée** et l'utilisateur ne pourra pas la changer.*
      - Il ne sera même pas possible de spécifier l'attribut dans le fichier de données XML
  - **#IMPLIED** : définir un attribut facultatif
    - *attention, ce n'est pas la valeur qui est facultative (car fournie par défaut), mais bien l'utilisation de l'attribut.*

DTD : Définition de listes d'attributs

- **#REQUIRED** : valeur de l'attribut est à fournir obligatoirement
  - *Si un fichier de données contient un tag, pour lequel un attribut est obligatoire, et si aucune valeur n'est fournie alors le document générera un message d'erreur.*

```
<!ELEMENT DATE (#PCDATA) >
 <!ATTLIST DATE format (en | fr) #REQUIRED >
```

```
<!-- La donnée est manipulable, car on qualifie son format -->
<DATE format="fr">26/08/1973</DATE>
```

DTD : Définition de listes d'attributs

- **#FIXED :** forcer un attribut à une valeur donnée.
  - *l'attribut existe bien. Sa valeur est imposée et l'utilisateur ne pourra pas changer cette valeur. Il ne sera même pas possible de spécifier l'attribut dans le fichier de données XML*

```
<!-- Comme le tag PRE en HTML -->
 <!ATTLIST PRE xml:space (default | preserve) #FIXED      "preserve" >
```

DTD : Définition de listes d'attributs

- **#IMPLIED** : définir un attribut facultatif :
  - *attention, ce n'est pas la valeur qui est facultative (car fournie par défaut), mais bien l'utilisation de l'attribut.*
  - *Exemple : l'attribut alt du tag <IMG> en HTML.*
    - permet de définir le texte alternatif de l'image : il apparaît, sous forme d'info-bulle, si vous laisser la souris immobile sur l'image pendant quelques instants.
      - L'attribut est facultatif : s'il n'est pas fournit, aucune info-bulle n'apparaîtra (même pas une info-bulle sans texte).

```
<!ATTLIST MON-IMAGE
            alt CDATA #IMPLIED
>
```

XML Shema

- permettant de définir la structure d'un document XML.
- Une instance d'un XML Schema est un peu l'équivalent d'une DTD, MAIS
  - XML Schema est plus riche : ex. permet par exemple de définir des domaines de validité pour la valeur d'un champ
  - XML Schema est lui même un document XML, alors que DTD est doc SGML

```
<?xml version="1.0" encoding="UTF-8"?>
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="personne">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="nom" type="xs:string" />
     <xs:element name="prenom" type="xs:string" />
     <xs:element name="date_naissance" type="xs:date" />
     <xs:element name="etablissement" type="xs:string" />
     <xs:element name="num_tel" type="xs:string" />
    </xs:sequence>
   </xs:complexType>
  </xs:element>
 </xs:schema>
```

Objectifs généraux des schémas

- **XML schéma est l'analogue**
  - D'un **langage de définition de données DDL** des bases de données
    - *un document est l'analogue d'un article d 'une BD*
  - D'un **langage de syntaxe abstraite** des réseaux comme ASN1
    - *un doc est l'analogue du contenu en syntaxe de transfert (BER 'Basic Encoding Rules').*
  - D'un **langage évolué dans sa partie définition de données** comme C++ ou Java.
    - *Un schéma définit l'analogue d 'une classe et*
    - *un document est une instance de cette classe.*

- Un document XML **doit pouvoir être validé** relativement à son schéma

XML Shema: Objectifs

1. - Structures –

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

2. - 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

Typage des données

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 déclaration**
  - Déclaration d'***éléments.***
  - Déclaration d'***attributs.***

- **Composants de définition de types**
  - *Définition de types* ***simples*** ***(Simple type).***
  - *Définition de types* ***complexes*** ***(Complex type).***

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 2 attributs optionnels :
  - **Use**
  - **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

- 'SimpleType' : ==> définir des éléments ou 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 : complexType

- 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" minOccurs="0" />
      <xsd:element name="items"  type="Items" maxOccurs= "30" />
   </xsd:all>
</xsd:complexType>
```

Structure

Les types de données XML schéma

Objectifs de la définition des types

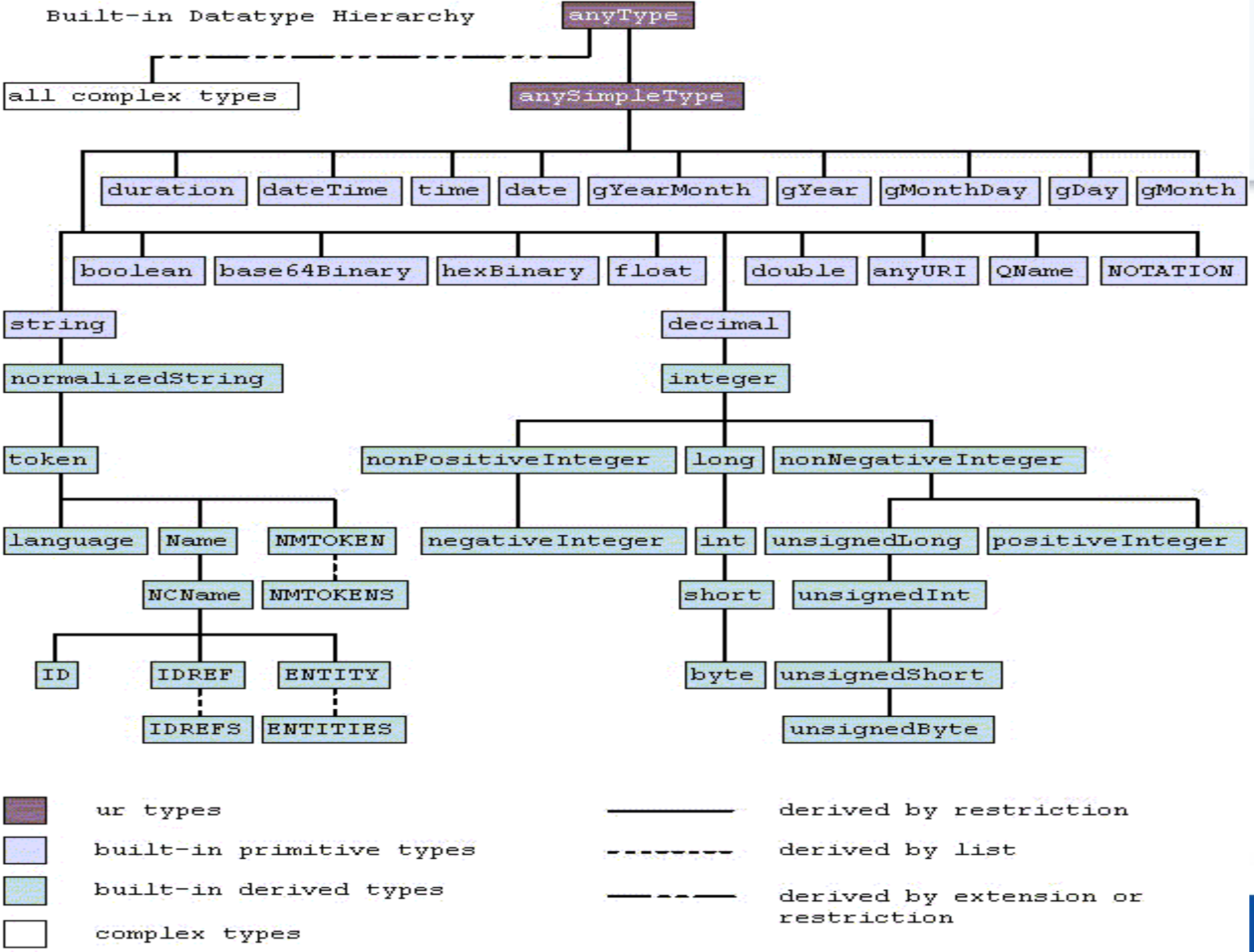
- **Fournir des types primitifs** analogues à ceux qui existent en SQL ou Java
- **Définir un système de typage suffisamment riche** pour
  - importer/exporter des données d'une base de données
- **Distinguer les aspects reliés à la représentation lexicale des données** de ceux gouvernant les ensembles 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
  - **L'ensemble des valeurs du type ('value space')**
    - *Ex*: type float
  - **L'ensemble des représentations lexicales possibles** des valeurs ('lexical space').
    - *Ex: "*10" *ou* "1.0E1"
  - **L'ensemble des facettes** *(l'ensemble des propriétés)* qui définit l'ensemble des valeurs
    - *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.

Définitions relatives aux types

- **Types primitifs** (‘Primitive’) Non défini en référence à d’autres types.
- **Types dérivés** (‘Derived’) Définis par dérivation à partir d’autres types
- **Types prédéfinis** (‘Built-in’) Définis dans le cadre de la spécification XML Schéma datatypes (primitif ou dérivé).
- **Types usagers** (‘User-derived’) Types construits par les utilisateurs.
- **Types atomiques** (‘Atomic’) Types indivisibles du point de vue de la spécification XML schéma.
- **Types listes** (‘List’) Types dont les valeurs sont des listes de valeurs de types atomiques.
- **Types unions** (‘Union’) Types dont les ensembles de valeur sont la réunion d’ensemble de valeurs d’autres types.


Built-in Datatype Hierarchy
anyType
all complex types
anySimpleType
duration
dateTime
time
date
gYearMonth
gYear
gMonthDay
gDay
gMonth
boolean
base64Binary
hexBinary
float
double
anyURI
QName
NOTATION
string
decimal
normalizedString
integer
token
nonPositiveInteger
long
nonNegativeInteger
language
Name
NMTOKEN
negativeInteger
int
unsignedLong
positiveInteger
NCName
NMTOKENS
short
unsignedInt
ID
IDREF
ENTITY
byte
unsignedShort
IDREFS
ENTITIES
unsignedByte
ur types
built-in primitive types
built-in derived types
complex types
derived by restriction
derived by list
derived by extension or restriction

Quelques types prédéfinis

Type	Forme lexicale
String	**Bonjour**
boolean	{**true**, **false**, **1**, **0**}
float	**2345E3**
double	**23.456789E3**
decimal	**808.1**
dateTime	**1999-05-31T13:20:00-05:00.**
binary	**0100**
uriReference	**http://www.cnam.fr**

….

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>
```

Dérivation de types simples
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>
```

Dérivation de types simples
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>
```

Dérivation de types simples
4 - Dérivation par liste

Une **liste** permet de définir un nouveau type de sorte qu'une valeur du nouveau type est une liste de valeurs du type pré existant (valeurs séparées par espace).

Exemple

```
<simpleType name= 'DebitsPossibles'>
        <list itemType='nonNegativeInteger'/>
</simpleType>
<debitsmodemV90 xsd:type='DebitsPossibles'>
33600 56000
 </debitsmodemV90>
```

Autres aspects XML : Définition d'entités internes

- ***Quoi*** ? Une entité est qualifiée d'interne si sa définition est directement embarquée dans la DTD.
- ***Pourquoi*** ? pouvoir remplacer autant de fois que nécessaire l'entité par le texte qui lui est associé.
  - *(similaire à #define de C)*

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE test [
        <!ENTITY titre "Introduction au langage XML" >
        <!-- . . . -->
]>
 <test>
        &titre;
</test>
```

Définition d'entités externes

- ***Pourquoi*** ? permet de substituer au nom de l'entité un contenu *localisé dans un autre fichier*.
- ***Exemple*** : Nous avons deux fichiers XML : chacun de ces fichiers définit un chapitre d'un livre.

Chap1.xml

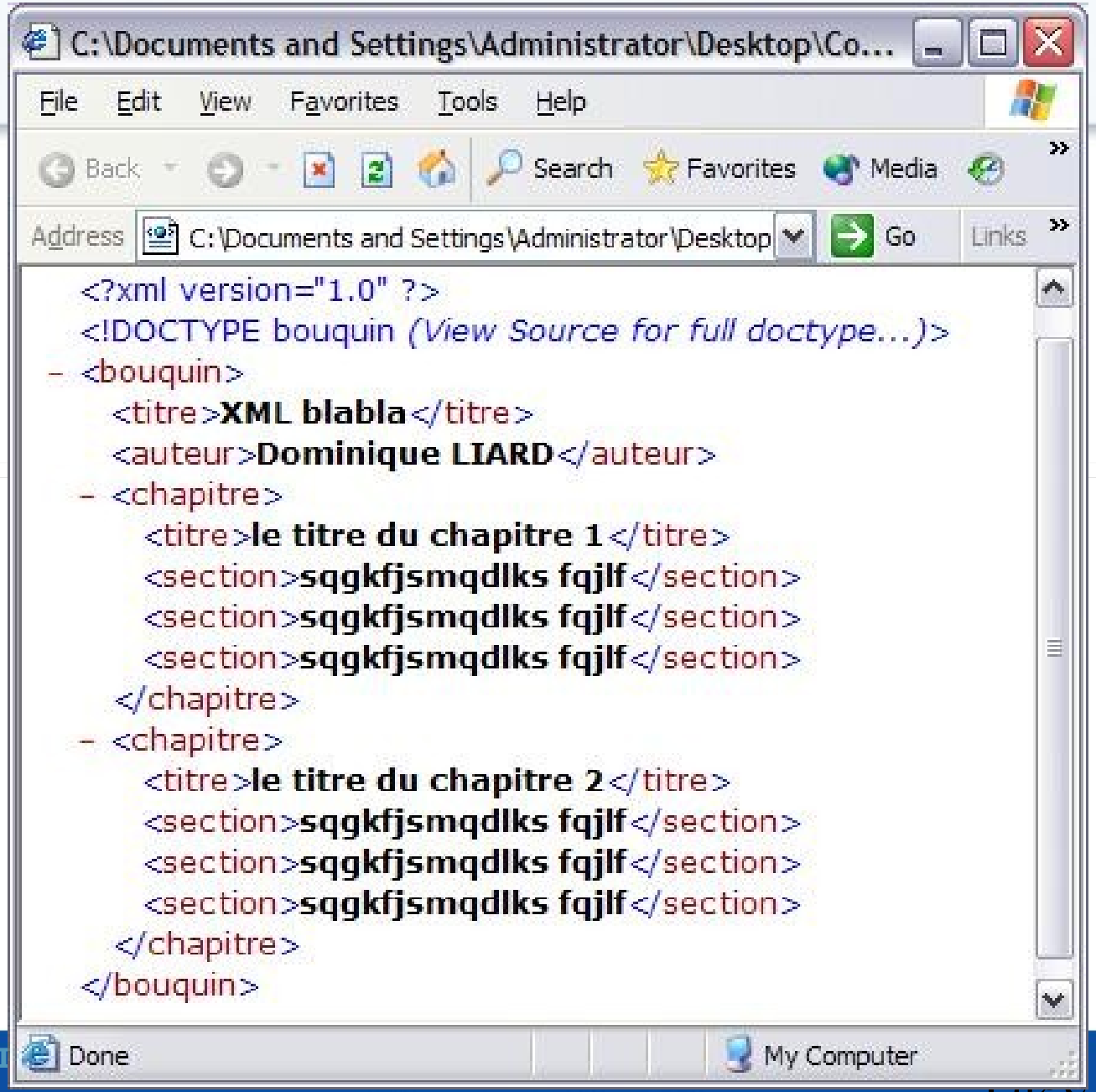
```
<?xml version="1.0" encoding="ISO-8859-1" ?
>
<chapitre>
      <titre>le titre du chapitre 1</titre>
      <section>sqgkfjsmqdlks fqjlf</section>
      <section>sqgkfjsmqdlks fqjlf</section>
</chapitre>
```

Définition d'entités externes

bouquin.xml

```
<?xml version="1.0" ?>
<!DOCTYPE bouquin [
        <!ENTITY chapitre-1 SYSTEM "./chapitre1.xml">
        <!ENTITY chapitre-2 SYSTEM "./chapitre2.xml">
        <!ENTITY auteur "Dominique LIARD">
]>
<bouquin>
        <titre>XML blabla</titre>
        <auteur>&auteur;</auteur>
        &chapitre-1;
        &chapitre-2;
</bouquin>
```

Définition d'entités externes



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.
- ...

XML s'impose …

- XML est en passe de devenir le standard utilisé par les e/ses pour échanger/stocker/traites... données dans des documents structurés,
  - que ce soit en interne,
  - avec des partenaires commerciaux ou
  - dans des applications accessibles à tous sur internet.

- Toutefois, les données échangées sont souvent stockées dans une BD relationnelles ou autre …
- Il faut donc mettre en place un processus de traduction des données du format actuel de la base en XML, puis du XML au format utilisé pour le traitement des données.

XML s'impose …

- Solution : utiliser certains des outils disponibles avec les BD relationnelles, comme Oracle9i et IBM DB2.
  - traduire en XML les données (en exécution), structurées ou non.
- Toutefois, cela risque d'augmenter les temps de traitement des transactions XML et de ralentir également les autres applications qui accèdent à la BD relationnelles.
- Les solutions relationnelles DB2 (IBM), SQL Server (Microsoft), Oracle 10g et ASE (Sybase) sont livrées en cas de besoins avec une sur-couche (moteur) permettant de générer des structures de données multidimensionnelles

- **Traduction = traitements plus longs**

- autre approche ? BD XML natives
  - Ces BD ne remplacent pas les BD relationnelles.
  - Elles servent plutôt de cache intermédiaire entre les applications web et les sources de données pour améliorer les performances.

Apports …

- accélérer le processus,
- Echange
  - traduire des données provenant de plusieurs sources en XML.
  - Ainsi, toutes les applications qui doivent échanger des données XML n'ont plus qu'une seule BD XML native à utiliser pour mettre les documents XML en cache.
    - *Ce qui réduit les besoins de traitement que requiert la gestion des traductions XML.*
- Les outils sont fournis par les BD compatibles XML permettent :
  - *d´interroger le document,*
  - *sélectionner les informations,*
  - *conserver et cataloguer les différentes versions de contenus, etc.*

Bases de données XML : Apports …

- *Stockage dans une BD XML permet d´effectuer des requêtes*
  - *Il est alors possible d´appliquer le principe de recherche et d'extraction d´informations à différentes activités telles que*
    - le travail collaboratif,
    - les échanges entre entreprises.
  - Catégorisés et classés, les détails d´un projet peuvent en effet être retrouvés plus facilement.
- Exemple
  - offrir différentes vues de ses informations générées à partir d´un document : titre et chapeau de l´article uniquement, totalité du document, parties relatives à un sujet spécifique demandé par le visiteur, etc.
    - *permet à un éditeur de dictionnaire généraliste, par exemple, d'extraire tous les mots relatifs à la botanique et sortir un dictionnaire spécifique.*

Bases de données XML : Apports …

- toute information extraite d´une base XML est structurée par son balisage, lequel peut être exploité par le destinataire
- Ce dernier peut :
  - automatiquement classer l'information dans sa propre base XML,
  - la publier sur son site ou encore dans une base de données relationnelle en s'appuyant sur des mécanismes normalisés de transformation des données pour adapter l'information à son propre format de publication ou de stockage.

Bases de données XML : Apports …

- *Non répudiation*
  - *stockage XML permet de respecter et conserver le document dans son intégralité*
  - *Si, par exemple, les informations d´une facture sont éclatées dans une base de données relationnelle, l´intégrité du document original est perdue. Il n´est donc plus exploitable en cas de litige*".
  - BD XML : sont dotées de fonctions de gestion des versions
    - *peuvent en outre cataloguer les évolutions d´un document et ainsi restituer les différentes étapes d´un projet ou d´une commande.*

Bases de données XML : Apports …

- outils de requête et de gestion des versions
  - ➔ gestion de contenu ou historisation de documents
  - MAIS : seule l´information balisée peut être restituée.
- *Lourd, ce travail de balisage* peut parfois être automatisé mais implique de profonds bouleversements dans les habitudes de travail des émetteurs de documents, des migrations lourdes en termes de compétences, de solutions techniques, etc.
- Avant de basculer en XML, de bien peser les implications, le stockage en XML constituant finalement la partie la plus simple d´un tel projet.

Les principales bases XML du marché

- La majorité des bases relationnelles du marché - Oracle 9i d´Oracle, Sybase ASE de Sybase, SQL Server de Microsoft, DB2 d´IBM - disposent de fonctions plus ou moins élaborées pour gérer et exploiter les documents en XML.
- En parallèle de ces offres, il existe des bases de données "natives" XML, conçues spécifiquement pour ne gérer que des contenus XML:
  - Software AG (Tamino),
  - Ixiasoft (TextML),
  - Ipedo (XML Database), etc.

SGBDR et XML : le tour de l'offre

Editeur/Nom	*OS*	*Développement*	*Sécurité*	*XML*
IBM DB2	Linux, UNIX, Windows	C++, Cobol, JDBC et PHP	Chiffrement : triple DES et DEA Authentification (LDAP)	Traduction des documents XML pour les intégrer à la structure relationnelle de la base.
Oracle Oracle10g	Linux, Mac OS X, UNIX et Windows	PLSQL, JDBC et C++	Chiffrement : triple DES et AES. Authentification (LDAP)	Intégration de contenus XML, description de la structure au format XML et, à l'inverse, des données XML sous forme relationnelle
SQL Server Microsoft	Windows	ODBC, C++, Visual Basic, .Net	Chiffrement : RC4, DES, PCT et SSL. Authentification (ActiveDirectory)	Support des données XML de manière native et définition de tables dans le même format.
Sybase ASE	ac OS X, UNIX, Windows.	ODBC, JDBC, C++, Perl et PHP	Chiffrement : SSL. Authentification (LDAP)	Stocke et indexe les fichiers XML. Support des requêtes au format XQL lancées sur cette catégorie de contenu

BD relationnelles Open Source et XML : le tour de l'offre

Editeur/Nom	*OS*	*Développement*	*Sécurité*	*XML*
Interbase	Linux, UNIX, Windows	C++ et Java	Authentification	Stocke et indexe les fichiers XML.
MySQL	Linux, Mac OS X, UNIX et Windows	ODBC, JDBC, C++, Java, Perl, PHP et Ruby	Chiffrement : SSL. Authentification et identification (X509)	Stocke et indexe les fichiers XML.
PostgreSQL	Linux, Mac OS X, Unix, et Windows	ODBC, JDBC, PL/SQL, C++, Perl, PHP, TLC et Ruby	Chiffrement : SSL. Authentification (HTTP).	Stocke et indexe les fichiers XML. Support des fonctions XPath.
MaxDB	Linux, Unix, et Windows	ODBC, C++, JDBC, Perl et PHP	Chiffrement : SSL. Authentification (HTTP).	Stocke et indexe les fichiers XML.

BD XML : le tour de l'offre

Editeur/Nom	*OS*	*Développement*	*Sécurité*	*XML*
Ipedo XML Database	Linux, UNIX, Windows	COM, Java et SOAP. Support des requêtes XPath et XQuery		Gestion de contenu XML par le biais d'un système de catégorisation
IXiasoft TextML	Windows	ODBC, JDBC, C++, Java, Perl, PHP et Ruby		L'outil s'appuie sur le "parser" d'IBM pour indexer l'ensemble des attributs XML sous forme de meta-données
Software AG Tamino	Linux, UNIX et Windows	ODBC, JDBC, C++ et COM. Support des requêtes XPath et XQuery	Chiffrement : JCE, Bouncy Castle et FlexSecure. Authentification (XKMS)	Combine une base de données XML à une base de données relationnelle.
XPEERiON **Agilience**	Windows	Java. Support des requêtes XPath et XQuery		Un connecteur pour .Net est en préparation. Agilience devrait être prochainement porté sous Linux

BD XML Open Source : le tour de l'offre

Editeur/Nom	*OS*	*Développement*	*Sécurité*	*XML*
Apache Xindice	Linux, MacOS X et Windows	Java. Support de SOAP et des requêtes XQuery et XML:DB		Stocke et indexe des fichiers XML ou non au sein d'une structure dans le même format.
eXist	Linux et Windows.	Java. Support de SOAP et des requêtes XQuery et XPath	Authentification (Cocoon).	Stocke et indexe des fichiers XML au sein d'une structure dans le même format

Galax : open-source implementation of XQuery (et XPath)

Bases de données multidimensionnelles

Editeur/Nom	*OS*	*Développement*	*Sécurité*	*XML*
Hyperion Essbase	UNIX et Windows	OLAP. Java, C++ et Visual Basic. Support des requêtes XML MaxL DML	Chiffrement SSL. Authentification (LDAP)	Les connecteurs pour Essbase sont nombreux
NCR Teradata Database	UNIX et Windows.	OLAP. Java et CGI.	Authentification (LDAP)	Les solutions qui accèdent à Teradata sont a priori moins nombreuses que pour Essbase
SAP SAP BW	Linux, UNIX et Windows	OLAP. BAPI (API SAP) et XML.	Authentification (LDAP).	Dédié à la plate-forme SAP. Il peut être attaqué par des applications tierces (via ODBC).
SAS Intelligence Storage	UNIX et Windows	OLAP. Java, JDBC et ODBC	Authentification (SAP et LDAP)	Associe des fonctions de structuration relationnelle et multidimensionnelle. Il peut être livré avec un ETL.

BD relationnelle ou BD XML : que choisir ?

- sources de données qui peuvent nécessiter traduction XML ?
- Si vous avez plusieurs sources de données à utiliser pour des applications XML ➔ une BD XML native est intéressante
  - Les BD XML intègrent des API que vous pouvez utiliser pour intégrer les données XML avec les applications
- Exemple
  - vous échanger des données avec d'autres sociétés qui utilisent XML pour gérer vos processus de facturation et de paiement
    - *Ces données peuvent être stockées dans une seule base de données*
  - en même temps vous échanger des données XML avec d'autres sociétés pour gérer les commandes en ligne et les livraisons.
    - *Ces applis peuvent utiliser une BD complètement différente, ou même des données stockées dans votre système de fichiers*
- Une BD XML native, mise en place au centre du dispositif avec un serveur d'applications le plus souvent, vous permettrait de traiter tous les processus XML de commandes, de paiement, de facturation et de livraison sans le moindre impact sur vos BD, les performances de votre système de fichiers ou les performances de vos applications web.

Une pour toutes

- Les BD XML natives permettent d'accéder aux autres sources de données en utilisant les méthodes standardisées d'accès aux BD
  - JDBC, ODBC.
  - Pratiquement toutes les BD, de FileMaker, MySQL, et PostgreSQL à Oracle, DB2 et Sybase, sont accessibles en source ou en cible pour les traductions XML.

désavantages

- technologie récente qui n'a pas encore été largement testée.
- très chères, même si on trouve des BD XML natives à des prix moins prohibitifs (open-source)
- assez faciles à installer
  - Des techniciens seront néanmoins nécessaires pour intégrer votre BD XML, vos sources de données et vos applications.
  - Un développeur expérimenté peut terminer l'intégration rapidement, mais il lui faudra être familier avec les API et les liaisons.
  - Un expert sera également requis pour configurer et administrer le tout de façon optimale.

Conseils

- Si votre société prévoit d'utiliser le XML à grande échelle ou à long terme, ces inconvénients ne sont pas si importants comparés aux avantages : performances améliorées, standardisation du XML et automatisation des processus commerciaux, **services Web**.

- Avant d'investir dans des technologies de BD XML natives, testez les capacités XML de vos BD existantes.
  - Il est probable que votre BD pourra traduire les données existantes en structures de documents XML.
  - Vous devrez alors mesurer le temps nécessaire à cette traduction.
- Si vous utilisez plusieurs sources de données (BD relationnelles fournies par différents développeurs, BD OO, données stockées dans systèmes de fichiers), vous devez vérifier si l'une des bases de données permet de traduire les données des autres en XML.
  - Il vaut mieux éviter d'avoir plusieurs bases de données qui traduisent en XML, car cela allonge les temps de traitement.

- La plupart des bases de données XML natives peuvent être téléchargées en version d'évaluation.
  - En essayer plusieurs avant de choisir celle qui semble la plus adaptée pour votre société

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/