

XML

Les bases en pratique

Support de cours réalisé par Joël FARVAULT (CNRS/DSI) et David ROUSSE (CNRS/DSI), relu par René PELFRESNE (CNRS/DSI)



Ce(tte) oeuvre est mise à disposition selon les termes de la Licence Creative Commons Paternité - Pas d'Utilisation Commerciale - Pas de Modification 2.0 France.

Avril 2003

Direction des systèmes d'information

- Tous les exemples présentés peuvent être obtenus par mail : david.rousse@dsi.cnrs.fr
- L'ensemble des exemples présentés dans les parties Transformations de documents XML, Liaisons de documents et Manipulations de documents XML a été testé sur la configuration suivante :
 - Windows XP SP1
 - Sun JDK 1.4.0_01 (installé dans C:\j2sdk1.4.0_01)
 - Apache Xerces 2.3.0 (installé dans C:\j2sdk1.4.0_01\xerces-2_3_0)
 - Apache Xalan 2.4.1 (installé dans C:\j2sdk1.4.0_01\xalan-j_2_4_1 mais copie xalan.jar de dans C:\Program Files\Java\j2re1.4.0_01\lib\endorsed)
 - Apache FOP 0.20.4 (installé dans C:\j2sdk1.4.0_01\fop-0.20.4)
 - PATH=%PATH%;C:\j2sdk1.4.0_01\bin
 - CLASSPATH=C:\j2sdk1.4.0_01\xerces-2_3_0\xercesSamples.jar;C:\j2sdk1.4.0_01\xerces-2_3_0\xercesImpl.jar;C:\j2sdk1.4.0_01\xerces-2_3_0\xml-apis.jar;C:\j2sdk1.4.0_01\xerces-2_3_0\xmlParserAPIs.jar;C:\j2sdk1.4.0_01\fop-0.20.4\build\fop.jar;C:\j2sdk1.4.0_01\fop-0.20.4\lib\batik.jar;C:\j2sdk1.4.0_01\fop-0.20.4\lib\avalon-framework-cvs-20020315.jar;C:\j2sdk1.4.0_01\fop-0.20.4\lib\logkit.jar;C:\j2sdk1.4.0_01\fop-0.20.4\lib\jimi-1.0.jar;
 - JAVA_HOME=C:\j2sdk1.4.0_01
- Il est possible de vérifier votre configuration ainsi :
 - lancer pour tester Xalan : java org.apache.xalan.xslt.EnvironmentCheck
 - lancer pour tester Xalan (Process) : java org.apache.xalan.xslt.Process -v
 - lancer pour tester FOP : java org.apache.fop.apps.Fop

-
1. Historique
 2. Structure et organisation du langage
 3. DTD et schémas XML
 4. Transformations de documents XML
 5. Liaisons de documents en XML
 6. Manipulations de documents XML
 7. Références

- 2 ateliers seront réalisés, pour la partie Transformations.

XML

Les bases en pratique

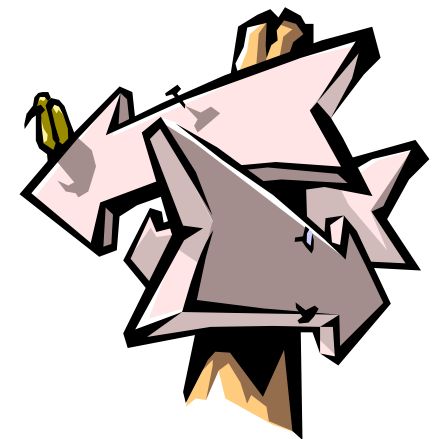
Support de cours réalisé par Joël FARVAULT (CNRS/DSI) et David ROUSSE (CNRS/DSI), relu par René PELFRESNE (CNRS/DSI)



Ce(tte) oeuvre est mise à disposition selon les termes de la Licence Creative Commons Paternité -
Pas d'Utilisation Commerciale - Pas de Modification 2.0 France.

Sommaire

1. Historique
2. Structure et organisation du langage
3. DTD et schémas XML
4. Transformations de documents XML
5. Liaisons de documents en XML
6. Manipulations de documents XML
7. Références



Historique

Historique – SGML et ses descendants

- La croissance d'Internet ces dernières années est à associer à la naissance de standards tels que :
 - HTML (*HyperText Marking up Language*)
 - HTTP (*HyperText Transfer Protocol*)
- A l'origine, un document HTML était constitué à part égale entre les données et la présentation. Mais les évolutions successives de HTML ont laissé la présentation prendre plus en plus de place dans un document HTML
- Il était nécessaire de définir un langage centré sur la description des données



Historique – SGML et ses descendants

- **Il est apparu aussi que l'échange d'information entre entreprises nécessite une définition et une description des données échangées**
- **Une première réponse à ces besoins a été l'élaboration de normes telles *EDIFACT* sous l'égide de l'ONU et le *SGML (Standardized Generalized Markup Language – norme ISO 8879)***
 - **EDIFACT a été utilisé dans le domaine des Echange de Données Informatisées (EDI)**
 - **SGML a été particulièrement utilisé dans la gestion documentaire**
- **Le point commun entre ces deux normes est leur complexité**

Historique – SGML et ses descendants

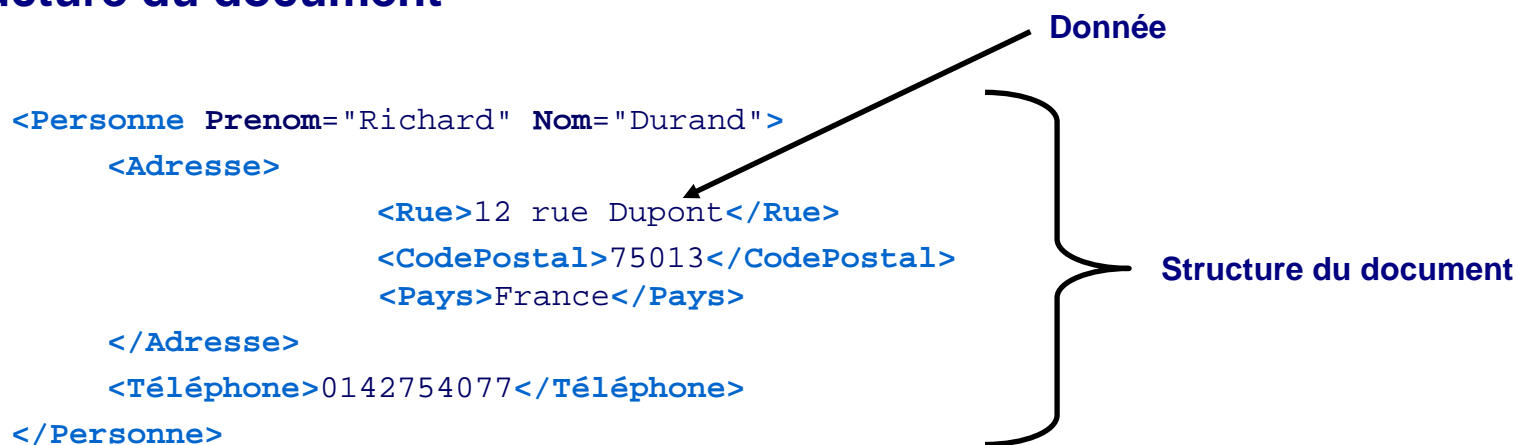
- En 1996 le W3C (*World Wide Web Consortium*) ainsi que plusieurs industriels ont travaillé sur la simplification de la norme SGML



- On retrouve dans la forme simplifiée du SGML les principes fondamentaux de ce langage :
 - Séparation de la présentation et des données
 - Structuration des données manipulées
 - Capacité de décrire les structures d'information à partir de schéma (DTD)
- Cette forme simplifiée du SGML a pris le nom de **XML** (eXtensible Marking up Language) en 1998

Historique – Définition de XML

- **Le XML (*eXtensible Marking up Language*) est un langage à balise définissant un format universel de représentation des données. Un document XML contient à la fois les données et des indications sur la structure du document**



- **XML est un format public, multi-plateforme, accessible à tous et supporté par des nombreux éditeurs de logiciels : IBM, Microsoft, Netscape, Oracle, Sun Microsystems...**

Historique – Définition de XML

- **XML n'est pas HTML... Pourquoi ?**
- **HTML présente plusieurs limitations :**
 - **HTML est orienté présentation et ne décrit pas la structure logique du document**
 - **Il est difficile d'associer des représentations physiques à un même document HTML (par exemple papier et écran)**
 - **Il n'est pas possible avec HTML, d'échanger des documents ou des données entre applications**
 - **Le langage HTML est construit sur des balises (mots placés entre < et >) prédéfinies dont très peu se rapportent à la structure du document**



Historique – Définition de XML

➤ En revanche XML...

- XML se présente sous forme d'arbre hiérarchique donc il décrit la structure logique du document
- XML n'utilise pas de balises prédéfinies, mais il propose des règles de construction. On peut inventer toutes les balises que l'on veut. C'est pour cette raison que le langage est dit *extensible*
- Pour être indépendant des évolutions technologiques, XML différencie : la structure, le contenu et la présentation



Historique – Présentation des champs d'application

- **XML promet de standardiser la manière dont l'information est :**
 - **Échangée**
 - **Adaptée**
 - **Présentée**
 - **Personnalisée**

- **C'est pourquoi les applications de XML se situent principalement sur :**
 - **La présentation des données**
 - **La portabilité des informations**

Historique – Présentation des champs d'application

➤ La présentation des données :

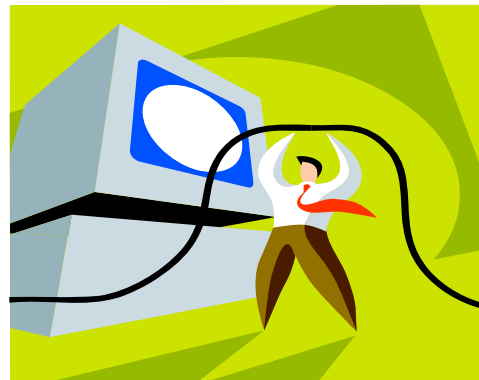
- XML permet de présenter les données en offrant une structuration adaptée à l'usage voulu. Dans le cas des sites Web, XML permet d'adapter les informations présentées à la personnalisation des interfaces des sites



- On peut saisir (ou mettre à jour) des informations à la manière dont on alimenterait une base de données, sans se soucier de la présentation ou même générer automatiquement la présentation (tableau, texte suivi...) sur de multiples médias

Historique – Présentation des champs d'application

- La portabilité des informations :
 - XML favorise les échanges informatisés ainsi que l'interopérabilité des applications. **La seule contrainte étant de définir un vocabulaire commun** (cf. référentiel partagé) détaillé dans des schémas (DTD et XSD). Par exemple le secteur de la chimie a défini un vocabulaire spécifique (CML) pour décrire les données du secteur chimique



Historique – Présentation des champs d'application

- **Les technologies et les domaines d'application utilisant XML sont :**
- **Développement des systèmes d'information**
 - **Administration et gestion de contenu**
 - **Multimédia (MPEG 7 est basée sur XML)**
 - **Gestion électronique de documents (GED) et portail d'informations d'entreprise**
 - **Intégration d'applications d'entreprise (EAI) avec la transformation des données vers XML (adaptateurs XML)**
 - **Commerce électronique et EDI**

Historique – Présentation des champs d'application

- **Les raisons pour lesquelles XML est utilisé de manière croissante :**
 - **Le XML remplace ASCII CSV pour l'échange de données**
 - **XML est indépendant du système cible ou d'un éditeur**
 - **XML devient le langage d'échange de données (utilisé par les EAI et les ERP)**
 - **Grâce à ses mécanismes de validation (DTD ou XSD) le XML favorise les échanges inter-applicatifs**
 - **XML supporte le dialogue client/serveur sur http et il devient le protocole unifié des architectures Web**
 - **XML est supporté par les SGBD majeurs et il est utilisé par Oracle pour décrire les méta-données**

Typologies des outils

➤ **Les outils liés à XML :**



- **Parseur XML (Apache Xerces, Microsoft MSXML)**
- **Processeur XSLT (Apache Xalan, Microsoft MSXML)**
- **Processeur XSL-FO (Apache FOP par exemple)**
- **Logiciel qui lit/écrit du XML (Microsoft Office 11, Open Office)**
- **Editeur XML (XML Spy, Cooktop ou XMLEdit)**

Structure et organisation du langage

Structure et organisation du langage – Syntaxe générale

- **Tout document présente une structure logique. Prenons le cas d'une lettre :**

En-tête

Monsieur Dupont
Compagnie des Bouts
à l'attention de M. DURAND

Apostrophe

Monsieur,

Corps

Suite à votre courrier du 30 Mars, nous accusons réception de votre courrier...
Nous vous prions d'agréer, Monsieur, l'expression de nos salutations distinguées

Signature

Monsieur Dupont

L'en-tête est suivie de l'apostrophe (Madame, Monsieur..) puis d'un corps et enfin une signature. C'est la structure logique de la lettre

Structure et organisation du langage – Syntaxe générale

- En revanche la structure physique du document est relative à sa *présentation* sur un support :



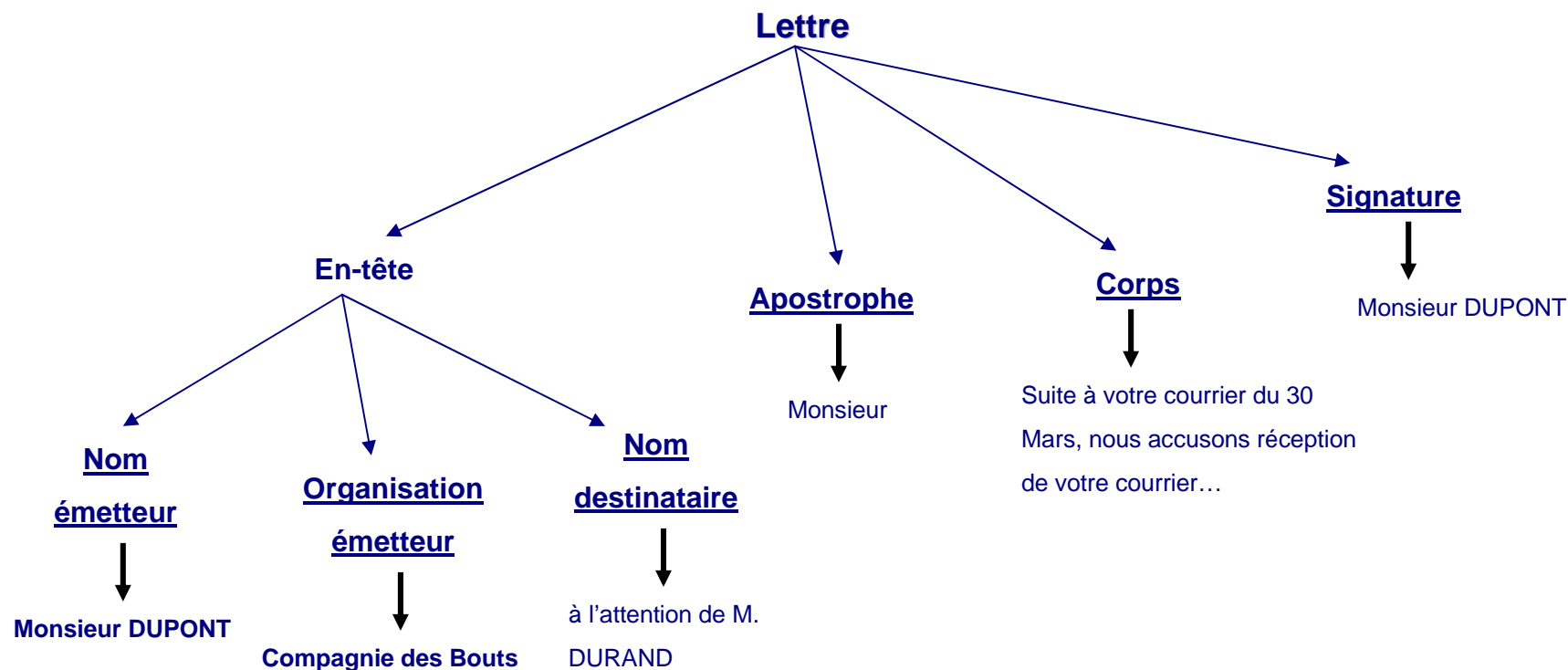
- Pour une présentation sur *papier* : la structure physique de la lettre précédente décrit que l'en-tête est placé en haut à gauche, que l'apostrophe est placée à trois centimètres au-dessous de l'adresse, que la première ligne des paragraphes est décalée de 1 cm à gauche...



- Pour une présentation sur *écran* : la structure physique du document pourra décrire que l'en-tête est en rouge et que le nom du destinataire clignote

Structure et organisation du langage – Syntaxe générale

- XML décrit la structure logique des informations, mais sous forme d'arbre. Pour une lettre, la structure logique décrite par XML sera :



Structure et organisation du langage – Syntaxe générale

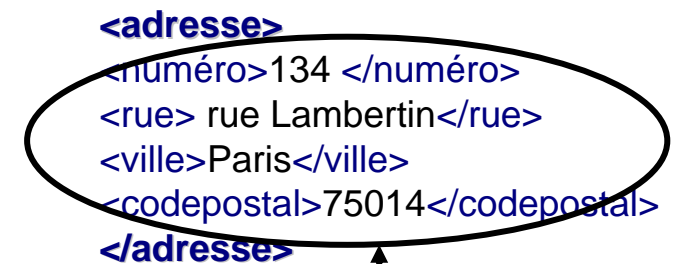
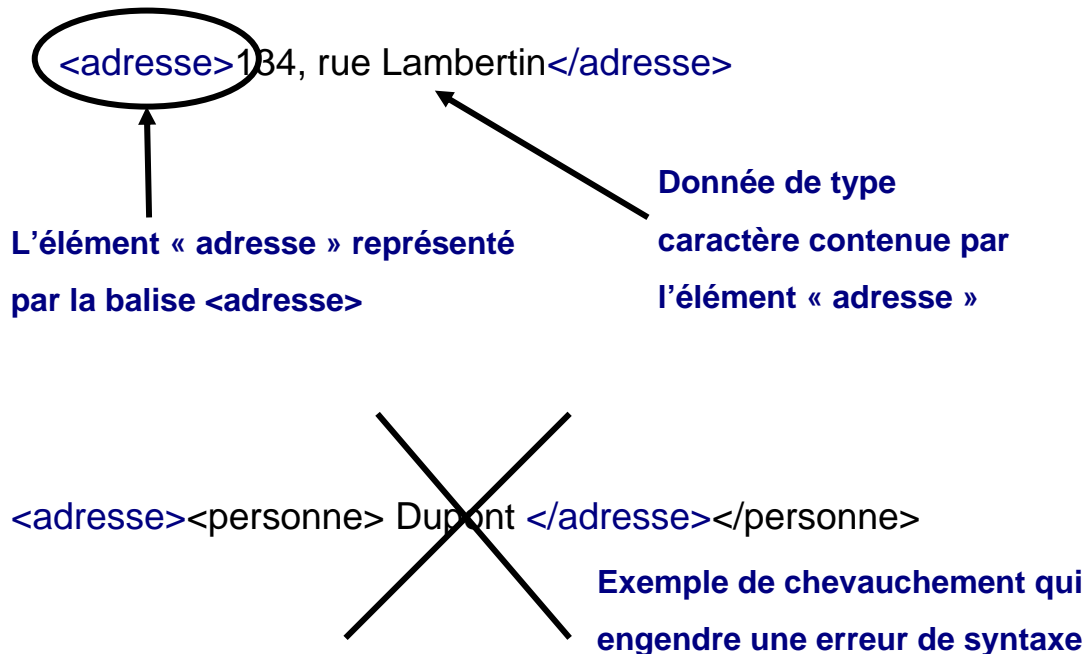
- On note la présence d'un élément racine « Lettre » auquel sont rattachés : « Corps », « Signature »...
- Un document XML commence toujours par un **élément « racine »** ou **élément document** qui se compose d'un nombre quelconque de sous-éléments imbriqués. Il contient toutes les données du document XML

```
<contact>  
<nom>Dupont</nom>  
<adresse>134, rue Lambertin</adresse>  
<ville>Paris</ville>  
<codepostal>75014</codepostal>  
</contact>
```

L'élément racine est <contact>

Structure et organisation du langage – Syntaxe générale

- Le document XML se compose d'**éléments** ce sont des balises qui contiennent des données analysables. Les éléments doivent s'imbriquer les uns dans les autres, aucun chevauchement n'est autorisé. **La donnée d'un élément ne peut contenir les caractères « & » ou « > »**



Structure et organisation du langage – Syntaxe générale

- Tout élément (y compris racine) d'un document XML est caractérisé par des **balises d'ouverture** et de **fermeture**

```
<contact>  
<nom> Dupont </nom>  
<adresse> 134, rue Lambertin</adresse>  
<ville> Paris</ville>  
<codepostal> 75014</codepostal>  
</contact>
```

Balise d'ouverture Balise de fermeture



Il est à noter que XML est sensible à la casse, donc le nom de l'élément dans la balise ouvrante doit être écrit de la même façon dans la balise fermante

Structure et organisation du langage – Syntaxe générale

- Un **élément vide** combine en une seule écriture une balise d'ouverture et de fermeture, car il ne contient pas de donnée

`<adresse />`

Cet élément ne contient aucune donnée mais dans sa notation il combine `<adresse>` et `</adresse>`

- Un élément peut posséder un (ou des) **attribut(s)** qui est un couple nom-valeur inclus à l'intérieur de la balise et délimité par des doubles (ou simples) quotes. **La donnée d'un attribut ne doit pas contenir les caractères « ^ », « % » ou « & »**

```
<Personne INSEE="172109254304523">
<Nom> DUPONT </Nom>
<Prenom> Jean </Prenom>
</Personne>
```

L'attribut INSEE est une caractéristique (numéro unique) de l'élément « Personne ». Un espace doit séparer le nom de l'élément et celui de l'attribut

Structure et organisation du langage – Syntaxe générale

- **Il est important de veiller à la présence des doubles ou simples quotes**

~~**<Personne INSEE=172109254304523>**
<Nom> DUPONT </Nom>
<Prenom> Jean </Prenom>
</Personne>~~

Cette notation est incorrecte

- **Un élément peut posséder plusieurs attributs qui doivent être séparés par des espaces**

<Personne INSEE="172109254304523" SEXE="Masculin" >
<Nom> DUPONT </Nom>
<Prenom> Jean </Prenom>
</Personne>

Structure et organisation du langage – Syntaxe générale

- Ces deux notations sont identiques :

```
<Personne INSEE="172109254304523" >  
<Nom> DUPONT </Nom>  
<Prenom> Jean </Prenom>  
</Personne>
```

=

```
<Personne>  
<INSEE> 172109254304523 </INSEE>  
<Nom> DUPONT </Nom>  
<Prenom> Jean </Prenom>  
</Personne>
```

- Mais il est préférable de limiter les attributs dans un document XML car ils :
- Ne décrivent pas une structure donc ils réduisent la clarté et la lisibilité du document XML
 - Sont plus difficilement manipulables avec XSL ou XSLT
 - Doivent être utilisés en tant que paramètres pour des éléments

Structure et organisation du langage – Syntaxe générale

- Les **données** portées par les éléments ou les attributs ne doivent pas contenir les caractères « ^ », « % », « & » ou « > »

- Le **nom des éléments et des attributs** doit aussi respecter des contraintes :
 - Il doit commencer par une lettre ou les caractères « _ » ou « - »
 - Il ne doit pas contenir d'« espace »
 - Il ne doit pas commencer par le nom « xml »



Atelier

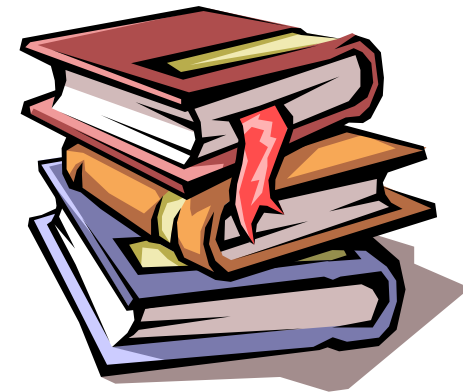
Création d'un document XML (1)



Structure et organisation du langage – Le prologue

➤ Le document XML peut contenir un *prologue* qui se compose de deux éléments :

- La déclaration XML
- La déclaration du type de document



➤ Le prologue ou l'en-tête du document XML n'est pas obligatoire, mais il contient des informations sur la version de XML, le jeu de caractère utilisé ainsi que les références du document

Structure et organisation du langage – Le prologue

- La **déclaration XML** apparaît au début du document et précise les paramètres auxquels se conforment le document. On définit en premier la **version de XML** (par défaut « 1.0 »)

`<?xml version="1.0" ?>`

On note la présence de délimiteurs spécifiques <? Et ?> qui sont à différencier de ceux habituels <...>

- Ensuite on précise **l'encodage des caractères**, c'est à dire la table de caractères utilisée par le document. « ISO-8859-1 » est couramment employée car les caractères accentués (« é » ou « è »...) seront acceptés dans le document XML

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

Utilisation de l'attribut encoding dans le prologue

Structure et organisation du langage – Le prologue

- La déclaration XML permet aussi de définir le document comme **autonome** c'est à dire qu'il est complet, aucun fichier externe (DTD) n'est référencé

```
<?xml version="1.0" encoding="ISO-5589-1" standalone="yes" ?>
```

Pour être autonome le paramètre standalone doit avoir la valeur « yes »

- En revanche s'il n'est pas autonome, il fait référence à une **DTD externe** contenant les « règles de grammaire » et les déclarations des balises utilisées dans le document XML. La DTD est renseignée par la **déclaration du type de document** (balise **DOCTYPE**)

```
<!DOCTYPE Adresse SYSTEM "http://document_type.fr/Adresse.dtd" >
```

Le mot-clé SYSTEM est un pointeur qui définit l'emplacement de la DTD

Structure et organisation du langage – Le prologue

- Un document XML contenant un prologue se présente selon cet exemple :

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE CARNETADRESSE [
  <!ELEMENT CARNETADRESSE (PERSONNE)*>
  <!ELEMENT PERSONNE (NOM,PRENOM,COMPAGNIE,EMAIL)>
  <!ELEMENT NOM (#PCDATA)>
  <!ELEMENT PRENOM (#PCDATA)>
  <!ELEMENT COMPAGNIE (#PCDATA)>
  <!ELEMENT EMAIL (#PCDATA)>
  <!ATTLIST PERSONNE ID ID #REQUIRED>
]>
<CARNETADRESSE>
  <PERSONNE ID="1">
    <NOM>Einstein</NOM>
    <PRENOM>Albert</PRENOM>
    <COMPAGNIE>Proton</COMPAGNIE>
    <EMAIL>ae@proton.org</EMAIL>
  </PERSONNE>
  ...
</CARNETADRESSE>
```

Ce prologue indique que l'on utilise la version 1.0 de XML, le document est autonome (DTD déclarée en interne)

Il est à noter que le *nom de l'élément racine* doit être identique à celui du DOCTYPE

Structure et organisation du langage – CDATA

- Pour remplacer certains caractères interdits (« & », « < » ..) on utilise les entités prédéfinies
- Mais si l'on souhaite insérer des données contenant des caractères interdits et pour lesquels il n'y aura *pas d'analyse syntaxique*, il faut utiliser la balise **CDATA** (*Characters DATA*). Tout le contenu compris dans la section `<![CDATA[` et `]]>` sera ignoré lors de l'analyse syntaxique du document XML

```
<script>  
  <![CDATA[  
    function swap(a,b) {  
      decl c;  
      c := a;  
      b := a;  
      b := c; }  
    ]]>  
</script>
```

Le contenu de CDATA ne sera pas pris en compte lors de l'analyse syntaxique

Structure et organisation du langage – Les commentaires

- On peut insérer des commentaires dans le document XML par les balises `<!--` et `-->`

`<!-- ceci est un commentaire -->`

- Il est à noter que :

- Un commentaire ne doit pas précéder le prologue

~~`<!-- ceci est un commentaire -->`
`<?xml version="1.0" ?>`~~

- Un commentaire ne peut être placé à l'intérieur d'une balise

~~`<VIDEO <!-- ceci est un commentaire --> >`~~

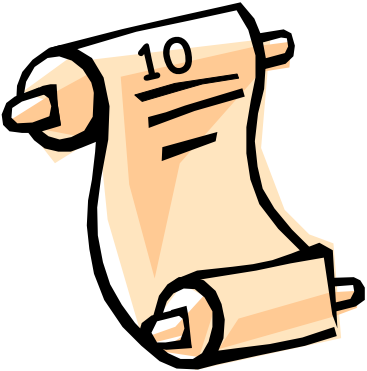
- Il ne doit pas avoir de « -- » dans le texte du commentaire
- Un commentaire ne doit pas apparaître dans une balise CDATA

Structure et organisation du langage – Le document bien formé

- Nous avons examiné ce qui constitue dans ses grandes lignes l'ensemble des règles de construction d'un document XML
- Le respect de ces règles de construction donnera lieu à un **document XML bien formé** qui ne peut être composé que de 6 types de balisages :
 - Les balises (ouvrantes, fermantes ou balises d'éléments vides)
 - Les références d'entités
 - Les commentaires
 - Les sections CDATA
 - Le prologue et les déclarations de type de document
 - Les instructions de traitement

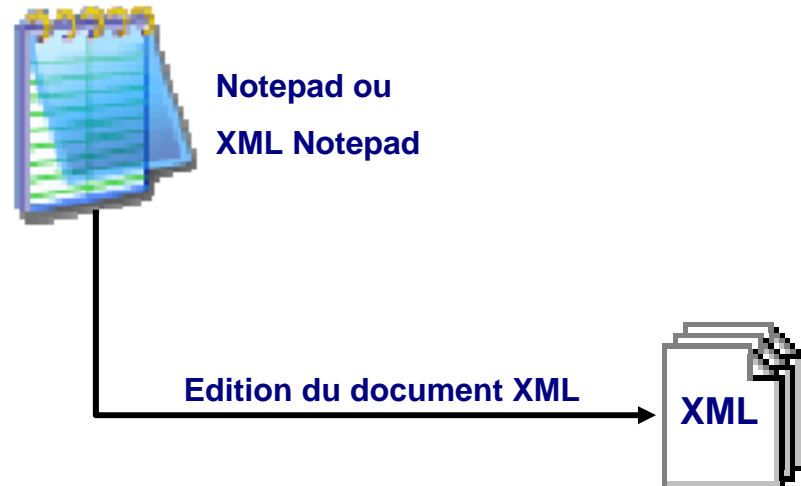
Structure et organisation du langage – Le document bien formé

- Un document ***bien formé*** doit aussi respecter ces critères de présentation :
- Un document XML ne peut être vide, il doit au moins contenir 1 élément
 - Il doit posséder un seul point de départ d'analyse, c'est-à-dire 1 seul élément racine
 - Il doit y avoir une imbrication correcte de ses éléments, c'est-à-dire un élément ne peut être fermé si tous les sous éléments qui le composent ne sont pas eux-mêmes fermés
 - Il doit contenir des références à des entités qui sont elles-mêmes bien formés



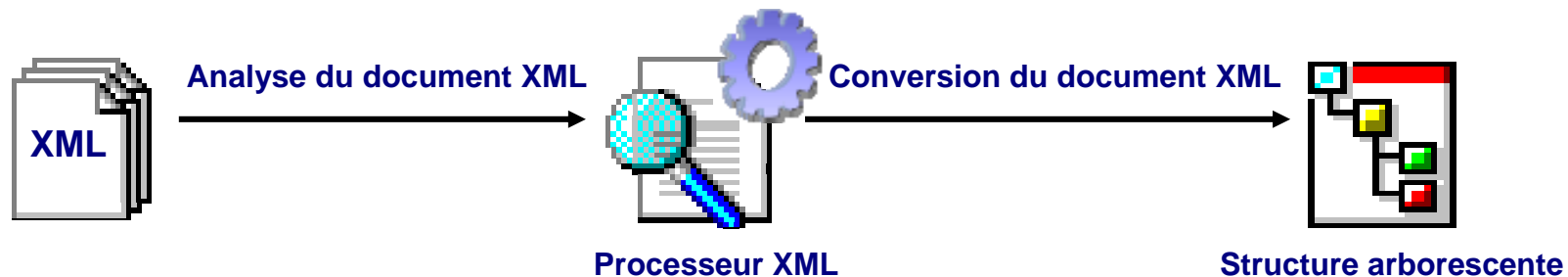
Structure et organisation du langage – Le document bien formé

- Un document XML est créé à partir d'un **éditeur XML** qui peut être un simple éditeur de texte (Notepad ou vi) ou un éditeur spécifique (XML Spy, XML Notepad..)



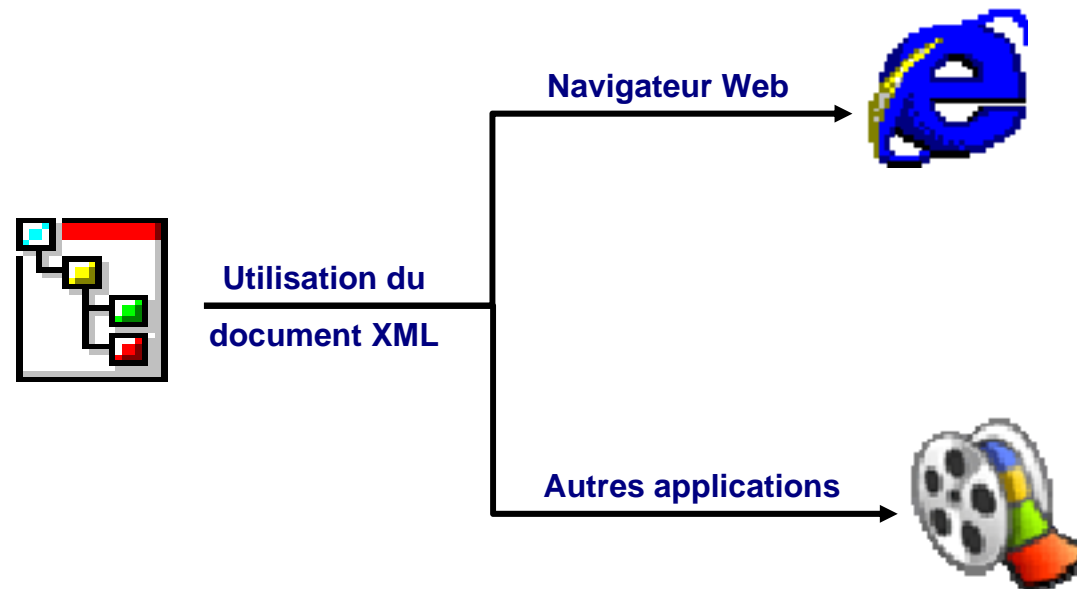
Structure et organisation du langage – Le document bien formé

- La vérification des règles de construction et l'interprétation du document est effectuée par un **analyseur XML** (appelé aussi parser, parseur ou processeur XML). Il va lire le document, vérifier qu'il est bien formé et le convertir en une structure d'éléments arborescents. Lors de cette phase d'analyse les entités seront remplacées par leur contenu



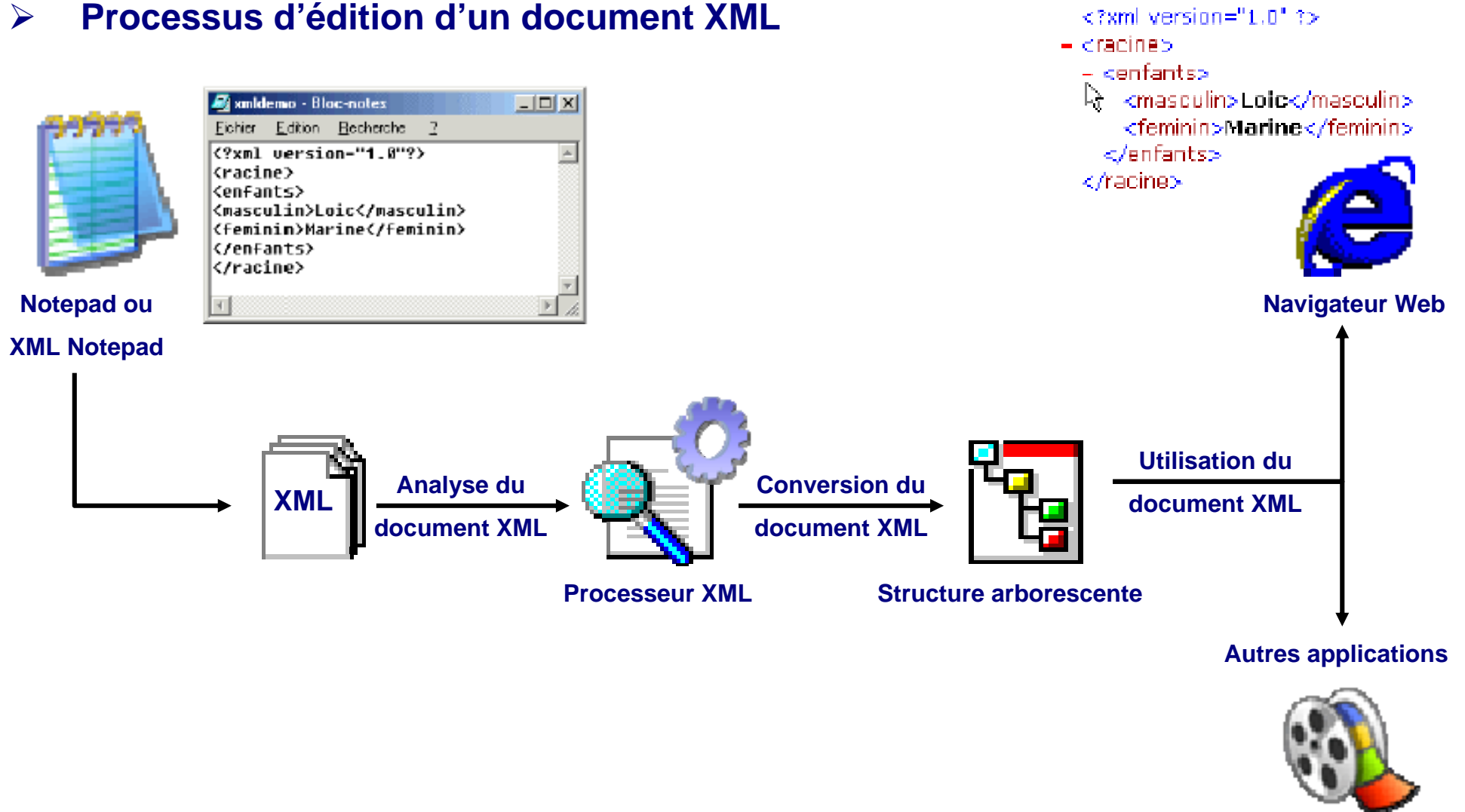
Structure et organisation du langage – Le document bien formé

- A la suite de ces tests de conformité, la structure arborescente ou certains nœuds (entités binaires) seront transmis à des **applications cibles**. Il peut s'agir d'un navigateur Web ou de tout autre programme capable de comprendre les données



Structure et organisation du langage – Le document bien formé

➤ Processus d'édition d'un document XML



Atelier

Création d'un document XML (2)



Structure et organisation du langage – Le document valide

- La notion de document bien formé est à distinguer de celle de document valide
- Un document est **bien formé** s'il est correct syntaxiquement et s'il répond aux exigences de construction d'un document XML
- Un document **valide** est un document bien formé dont les données et la structure sont conformes aux spécifications du document type (DTD)
- Qu'est-ce qu'une DTD ?



Les DTD et schémas XML

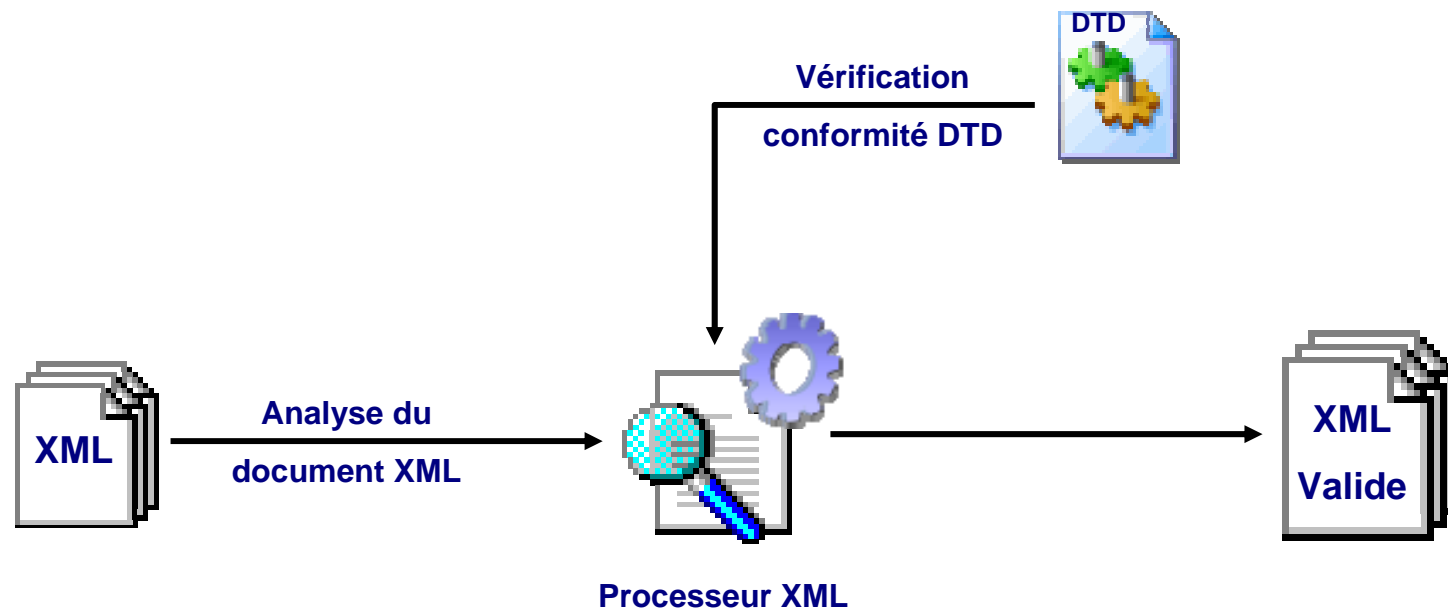
Les DTD et schémas XML – Les DTD

- Une **DTD** (*Document Type Definition*) est un modèle permettant de décrire la **grammaire** d'un document XML
- Une **grammaire** définit la syntaxe d'un langage, c'est-à-dire l'organisation de ces balises. Un document XML sera donc compréhensible si sa grammaire est définie
- La DTD définira aussi des contraintes portant sur la sémantique, la structure et les valeurs applicables pour un élément du document



Les DTD et schémas XML – Les DTD

- L'opération de vérification de conformité entre le document XML et la DTD est effectuée par le processeur (ou parseur) XML



Les DTD et schémas XML – Les DTD

➤ Pourquoi utiliser une DTD :

- C'est le moyen de déclarer de nouvelles balises et spécifier des contraintes sur ces balises
- C'est la possibilité pour une application de savoir quel document XML produire et quoi lire
- C'est la possibilité d'assurer l'interopérabilité entre applications en définissant des règles d'échanges communes
- C'est la possibilité de valider les flux XML entrants d'un programme

Les DTD et schémas XML – Les DTD

- Une DTD peut se présenter sous deux formes : *interne* ou *externe*
- **DTD interne** : la DTD est définie dans le document XML

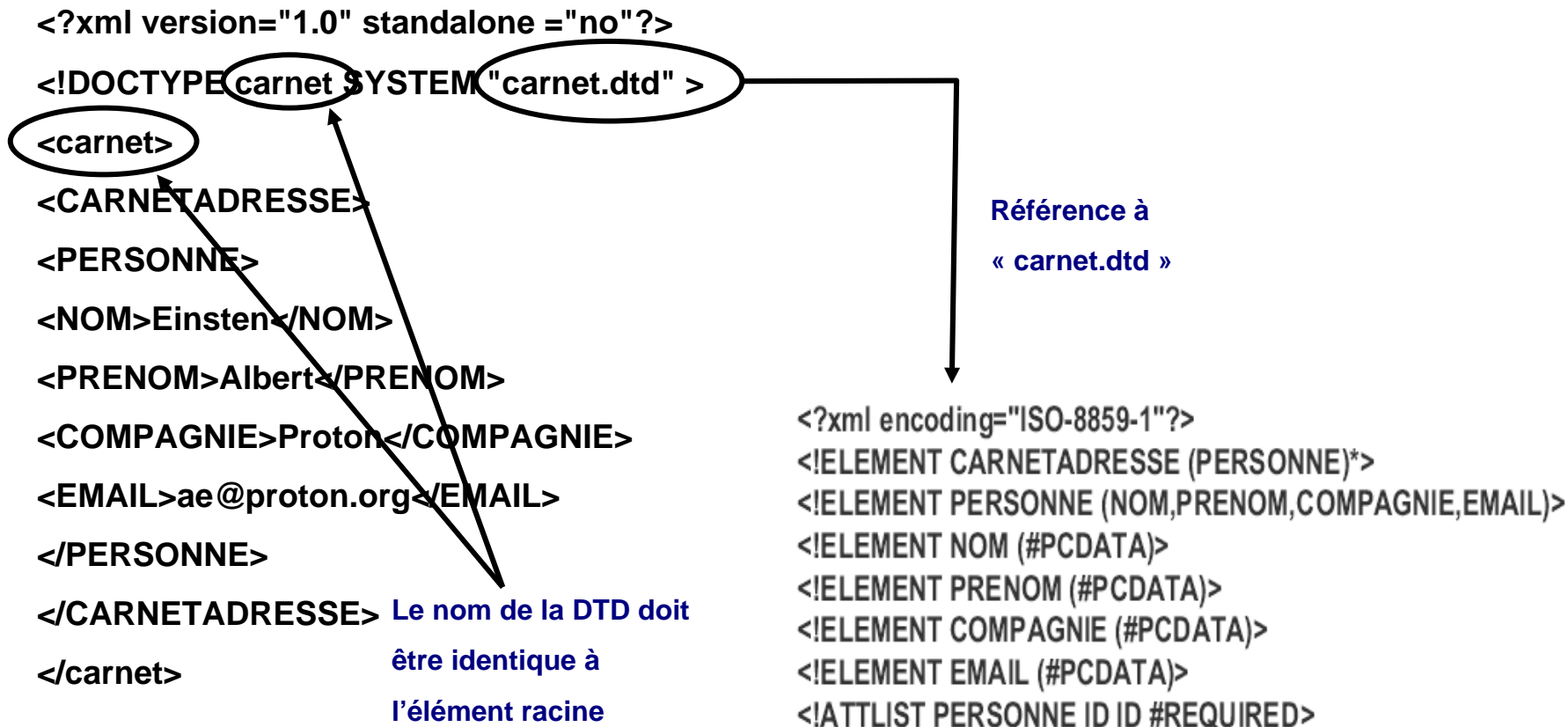
Les éléments placés entre crochets après la zone de déclaration du DOCTYPE constituent la DTD interne

```
<?xml version="1.0" standalone="yes"?>  
  <!DOCTYPE CARNETADRESSE [  
    <!ELEMENT CARNETADRESSE (PERSONNE)*>  
    <!ELEMENT PERSONNE (NOM,PRENOM,COMPAGNIE,EMAIL)>  
    <!ELEMENT NOM (#PCDATA)>  
    <!ELEMENT PRENOM (#PCDATA)>  
    <!ELEMENT COMPAGNIE (#PCDATA)>  
    <!ELEMENT EMAIL (#PCDATA)>  
    <!ATTLIST PERSONNE ID ID #REQUIRED>  
  ]>  
<CARNETADRESSE>  
  <PERSONNE ID="1">  
    <NOM>Einstein</NOM>  
    <PRENOM>Albert</PRENOM>  
    <COMPAGNIE>Proton</COMPAGNIE>  
    <EMAIL>ae@proton.org</EMAIL>  
  </PERSONNE>  
  ...  
</CARNETADRESSE>
```

Le nom de la DTD doit être identique à l'élément racine

Les DTD et schémas XML – Les DTD

- **DTD externe** : la DTD est référencée (Nom et URL) dans le prologue du document XML via la balise DOCTYPE



Les DTD et schémas XML – Les DTD

- Des références *internes* ou *externes* à des DTD peuvent figurer au sein d'un même document XML
- Les définitions des éléments et des attributs figurant dans les DTD internes et externes se complètent tant qu'il n'y a pas de doublons
- Toutefois, s'il y a des doublons dans plusieurs DTD, c'est *la DTD interne qui sera traitée prioritairement*

Les DTD et schémas XML – Les éléments

- La DTD contient la description des **éléments** et des **attributs** qui seront acceptés dans le document XML
- Une déclaration d'élément prend la forme :

<!ELEMENT [nom de l'élément] ([contenu de l'élément ou type])>

Exemple <!ELEMENT personne (nom, email)>

Correspond

```
<personne>
  <nom> ... </nom>
  <email> ... </email>
</personne>
```



Rappel : Le nom de l'élément ne doit pas contenir de caractère « espace », on peut utiliser les caractères « - » ou « _ »

Les DTD et schémas XML – Les éléments

- La déclaration de l'élément implique de définir le type de données qu'il contient, à choisir parmi 5 types :
 - Liste de sous éléments appelée « modèle de contenu » (cf exemple précédent)
 - Type « EMPTY »
 - Type « ANY »
 - #PCDATA
 - Contenu mixte

Les DTD et schémas XML – Les éléments

- Le type *modèle de contenu* indique les sous éléments composants l'élément déclaré

<!ELEMENT MESSAGE (DESTINATAIRE, EXPEDITEUR, OBJET, CORPS)>

- Pour chacun de ces sous éléments, une déclaration d'élément distincte doit apparaître à la suite de la DTD

<!ELEMENT MESSAGE (DESTINATAIRE, EXPEDITEUR, OBJET, CORPS)>

<!ELEMENT DESTINATAIRE (#PCDATA)>

<!ELEMENT EXPEDITEUR (#PCDATA)>

...

Déclaration distincte de
l'élément

Élément racine composé de
DESTINATAIRE,
EXPEDITEUR, OBJET et
CORPS

Les DTD et schémas XML – Les éléments

- Des symboles permettent de spécifier des contraintes (ordre d'apparition, nombre d'occurrences...) sur les sous éléments :

Symbole	Utilisation	Exemple	Signification
() et ,	Encadre un groupe d'éléments et identifie l'ordre d'apparition	<!ELEMENT PERSONNE (NOM, PRENOM, VILLE)>	L'élément <i>Personne</i> doit contenir la séquence NOM, PRENOM et VILLE dans l'ordre
	Définit un groupe d'alternative	<!ELEMENT PERSONNE (NOM PRENOM VILLE)>	L'élément <i>Personne</i> doit contenir NOM ou PRENOM ou VILLE (au moins 1 des trois). Toutefois l'ordre n'est pas obligatoire

Les DTD et schémas XML – Les éléments

Symbole	Utilisation	Exemple	Signification
?	Indique qu'un élément peut apparaître 0 ou 1 fois	<p><!ELEMENT PERSONNE (NOM, PRENOM, VILLE?)></p> <p>Ou</p> <p><!ELEMENT PERSONNE (PRENOM, VILLE)?></p>	<p>- L'élément <i>Personne</i> peut contenir VILLE et si c'est le cas VILLE ne peut qu'apparaître qu'une fois</p> <p>- Même signification pour PRENOM et VILLE</p>
*	Indique qu'un élément peut apparaître 0 ou plusieurs fois	<p><!ELEMENT PERSONNE (NOM, PRENOM, VILLE*)></p> <p>Ou</p> <p><!ELEMENT PERSONNE (PRENOM, VILLE)*></p>	<p>- L'élément <i>Personne</i> peut contenir VILLE et si c'est la cas VILLE peut apparaître plusieurs fois</p> <p>- Même signification pour PRENOM et VILLE</p>
+	Indique qu'un élément peut apparaître une ou plusieurs fois	<p><!ELEMENT PERSONNE (NOM, PRENOM, VILLE+)></p> <p>Ou</p> <p><!ELEMENT PERSONNE (PRENOM, VILLE)+></p>	<p>- L'élément <i>Personne</i> doit contenir au moins 1 fois VILLE</p> <p>- Même signification pour PRENOM et VILLE</p>

Les DTD et schémas XML – Les éléments

Symbole	Utilisation	Exemple	Signification
[Aucun symbole]	Indique qu'un élément doit apparaître une seule fois	<!ELEMENT PERSONNE (NOM, PRENOM, VILLE)>	- L'élément <i>Personne</i> doit contenir NOM, PRENOM et VILLE une seule fois

Les DTD et schémas XML – Les éléments

- Dans la déclaration du modèle de contenu on peut utiliser plusieurs types de contraintes :

```
<!DOCTYPE CARNETADRESSE [  
  <!ELEMENT CARNETADRESSE (PERSONNE)*>  
  <!ELEMENT PERSONNE  
    (NOM,PRENOM,COMPAGNIE,EMAIL)>  
  <!ELEMENT NOM (#PCDATA)>  
  <!ELEMENT PRENOM (#PCDATA)>  
  <!ELEMENT COMPAGNIE (#PCDATA)>  
  <!ELEMENT EMAIL (#PCDATA)>  
>
```

L'élément
CARNETADRESSE peut ne
pas contenir de PERSONNE
ou en contenir plusieurs

L'élément PERSONNE doit
contenir 1 seule fois le
NOM, PRENOM,
COMPAGNIE et EMAIL

Les DTD et schémas XML – Les éléments

- Le type « **EMPTY** » permet de définir un élément vide (ne contenant pas de donnée mais il peut porter des attributs)

<!ELEMENT adresse EMPTY>

Correspond en XML

<adresse/>

- Le type « **ANY** » indique que l'élément peut contenir n'importe quel contenu autorisé par la DTD et dans n'importe quel ordre

<!ELEMENT **adresse ANY**>

<!ELEMENT ville EMPTY>

Ou

Ou

<adresse>12 rue dupont</adresse>

<adresse><ville/></adresse>

<adresse/>

Les DTD et schémas XML – Les éléments

- Le type « **#PCDATA** » (*Parsed Characters Data*) indique que les caractères contenus dans l'élément seront analysés par le processeur XML. A différencier de **#CDATA** (*Characters Data*) qui permet d'ignorer le contenu de l'élément

<!ELEMENT MESSAGE (DESTINATAIRE, EXPEDITEUR, OBJET, CORPS)>

<!ELEMENT DESTINATAIRE (#PCDATA)>

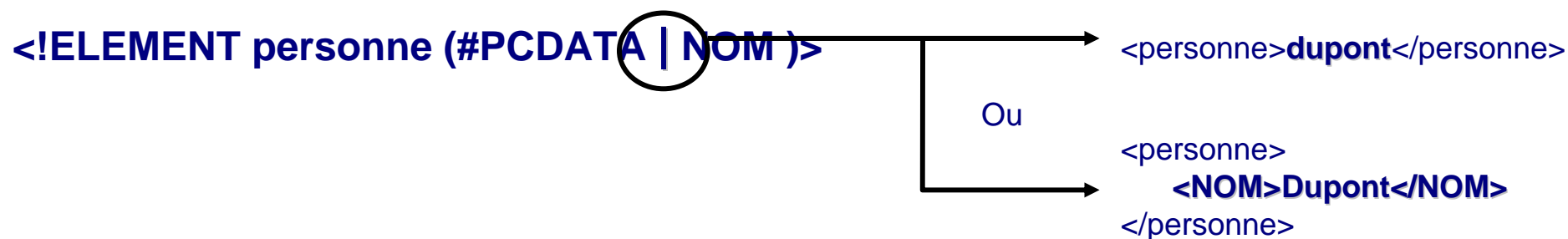
...

<DESTINATAIRE>Dupont Jean</DESTINATAIRE>

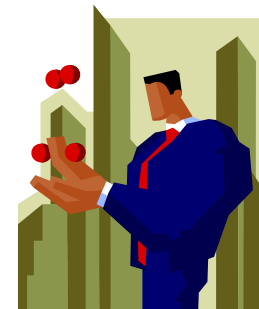
Indique que le contenu de l'élément DESTINATAIRE sera analysé par le processeur XML. Donc il ne peut contenir des caractères spéciaux tels que « & » ou « > »

Les DTD et schémas XML – Les éléments

- Le type **contenu mixte** « / » permet de choisir un type de contenu parmi une liste prédéfinie



Il est à noter que dans un *contenu mixte* #PCDATA doit toujours être déclaré en premier



Les DTD et schémas XML – L'entité

- Le document XML peut être composée d'**entités** qui correspondent à des « unités de stockage ». Chaque entité est identifiée par un nom et son contenu peut représenter un caractère aussi bien qu'un fichier externe
- La balise **entité** peut être utilisée pour des abréviations ou pour stocker des données autres que textuelles (images, vidéo...)
- L'utilisation de cette balise s'effectue en 2 étapes :
 - **Déclaration d'entités dans la DTD** : indique avec quoi il faut remplir l'entité
 - **Référence d'entités dans le document XML** : permet de retrouver le contenu de l'entité et l'utiliser à l'endroit souhaité

Les DTD et schémas XML – L'entité

- Pour définir une abréviation par exemple « XML », on **déclare une entité** (balise **ENTITY**) dans la DTD nommée XML auquel on associe la chaîne de caractère « eXtensible Marking up Language »

`<!ENTITY XML "eXtensible Marking up Language">`

L'entité se nomme XML
et contient la chaîne
« eXtensible Marking up
Language »

Attention : ce délimiteur commence par <! Mais
se termine par >

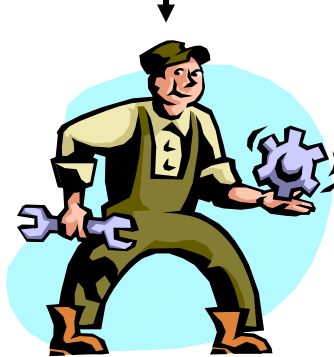
- Pour faire **référence à l'entité XML** dans le document XML, on utilise son nom précédé du symbole « & » et terminé par le point-virgule « ; » soit « &XML; »

`<LANGAGE> XML est &XML; </LANGAGE>`

Les DTD et schémas XML – L'entité

- La référence à l'entité sera ensuite interprétée et remplacée par son contenu

<LANGAGE> XML est **&XML;** </LANGAGE>



<LANGAGE> XML est **eXtensible Marking up Language** </LANGAGE>

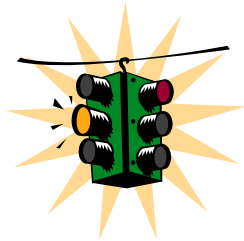
Les DTD et schémas XML – L'entité

- Il n'y a pas de limitation sur le nombre d'entités que l'on peut créer, toutefois il existe 5 entités prédéfinies :

<i>Entités</i>	<i>Caractères remplacés</i>
&lt;	<
&gt;	>
&amp;	&
&apos;	'
&quot;	«

Les DTD et schémas XML – L'entité

- On peut utiliser ces entités prédéfinies pour représenter des caractères interdits dans les données tels que « & », « > » ou « < »



`<instruction> if note < 10 </instruction>`

La notation est incorrecte car le caractère « < » n'est pas admis dans les données d'un élément

`<instruction> if note < 10 </instruction>`

En utilisant l'entité prédéfinie remplaçant le caractère « < » la notation est correcte

Les DTD et schémas XML – L'entité

- Les entités ne concernent pas seulement des abréviations mais aussi des déclarations de DTD. On parle *d'entités paramètres* ou *d'entités DTD*
- Les entités paramètres permettent de réutiliser des DTD existantes pour les enrichir ou ajouter de nouvelles spécifications

DTD 1

```
<?xml encoding="ISO-8859-1"?>  
<!ELEMENT CARNETADRESSE (PERSONNE)*>  
<!ELEMENT PERSONNE (NOM,PRENOM,COMPAGNIE,EMAIL)>  
<!ELEMENT NOM (#PCDATA)>  
<!ELEMENT PRENOM (#PCDATA)>  
<!ELEMENT COMPAGNIE (#PCDATA)>  
<!ELEMENT EMAIL (#PCDATA)>  
<!ATTLIST PERSONNE ID ID #REQUIRED>
```

DTD 2

```
<!ELEMENT ANNUAIRE (CARNETADRESSE*)>
```



Les DTD et schémas XML – L'entité

- Une entité paramètre se déclare dans la DTD :

<!ENTITY % [caractère « espace »] [nom entité] SYSTEM [emplacement de la DTD]>

- La référence à l'entité permet d'ajouter le contenu de l'entité dans la DTD, il s'effectue comme suit : **% [sans « espace »] [nom entité];**

Utilisation du caractère
« % » et non « & »

DTD 1

```
<?xml encoding="ISO-8859-1"?>  
<!ELEMENT CARNETADRESSE (PERSONNE)*>  
<!ELEMENT PERSONNE (NOM,PRENOM,COMPAGNIE,EMAIL)>  
<!ELEMENT NOM (#PCDATA)>  
<!ELEMENT PRENOM (#PCDATA)>  
<!ELEMENT COMPAGNIE (#PCDATA)>  
<!ELEMENT EMAIL (#PCDATA)>  
<!ATTLIST PERSONNE ID ID #REQUIRED>
```

DTD 2

```
<!ENTITY % CARNET SYSTEM « carnet.dtd » >  
..  
<!ELEMENT ANNUAIRE (CARNETADRESSE*)>  
%CARNET;
```

Référence à
« carnet.dtd »



Les DTD et schémas XML – L'entité

- La balise entité utilisée pour les abréviations ou les DTD est appelée **entité analysée**
- Mais il y a aussi l'entité non analysée souvent appelée **entité binaire** car elle contient des données binaires comme des images, des vidéos...
- Par exemple pour référencer un fichier image, on déclare une entité binaire à laquelle on associe l'emplacement du fichier et un format

<!ENTITY monimage(SYSTEM) "http://mesimages.fr/image1.gif"(NDATA GIF) >

Le mot-clé SYSTEM est un pointeur qui définit l'emplacement du contenu de l'entité

Le mot-clé NDATA signale que les données externes ne sont pas des données XML. En l'occurrence dans l'exemple, les données sont du type GIF

Les DTD et schémas XML – L'entité

- Pour faire référence à l'entité binaire (exemple : monimage) il faut la déclarer en tant que valeur d'attribut au sein d'un élément. Il faut indiquer le nom de l'entité (sans le « & » et « ; »)

```
<!NOTATION GIF SYSTEM "/usr/bin/gifview.exe" >
```

```
<!ENTITY monimage SYSTEM "http://mesimages.fr/image1.gif" NDATA GIF >
```

```
<collection>  
  <album photo="monimage"/>  
</collection>
```

```
<collection>  
  <album photo="&monimage;"/>  
</collection>
```

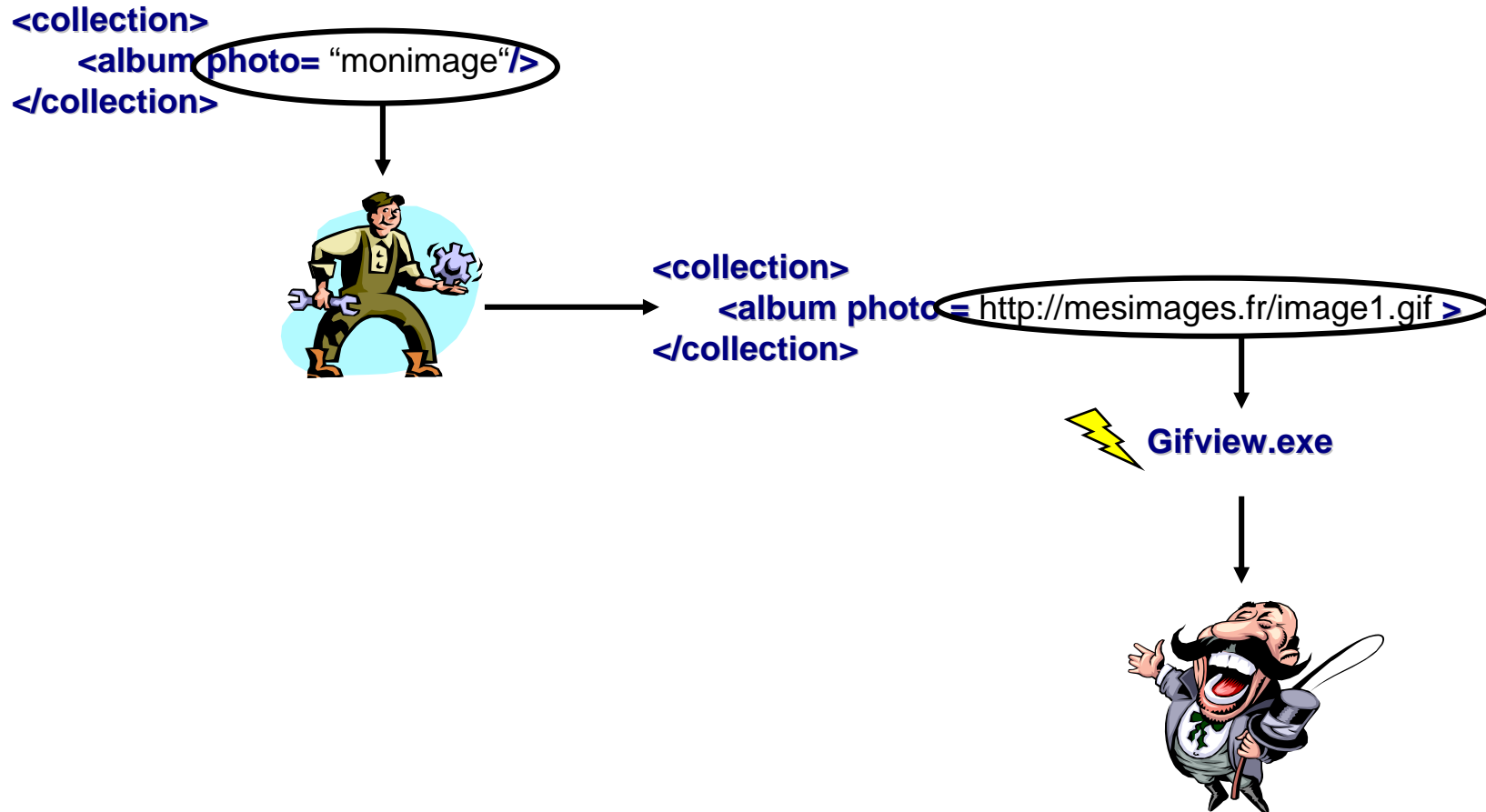
Déclaration de
l'entité binaire
« monimage » ainsi
que du format GIF

Cette notation est incorrecte

La valeur de l'attribut photo fait appel à l'entité binaire « monimage ». Noter que pour l'entité binaire, on utilise simplement le nom de l'entité et non la notation « &[nom entité];

Les DTD et schémas XML – L'entité

- La référence à l'entité binaire sera ensuite remplacée par son contenu et traitée par la commande de la balise **NOTATION**



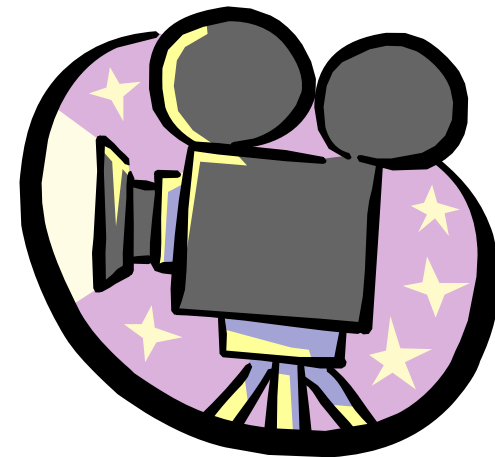
Les DTD et schémas XML – L'entité

- L'entité binaire peut être applicable sur tout type de fichier y compris des animations flash ou des vidéos

```
<!NOTATION flash SYSTEM "/usr/bin/flash.exe" >  
<!ENTITY animation SYSTEM "../anim fla" NDATA flash >
```

```
<scene loc="animation"/>
```

L'entité binaire « animation » provoquera la lecture de l'animation flash « anim fla »



Il est à noter que le document XML **ne doit contenir que l'appel ou le référencement de l'entité**

Les DTD et schémas XML – Les attributs

- Une déclaration d'attributs prend la forme :

`<!ATTLIST [nom de l'élément] [nom de l'attribut] [type] [options] [défaut]>`

Exemple `<!ELEMENT personne EMPTY>`

`<!ATTLIST personne prenom CDATA/>`

Tout comme l'élément, le nom de l'attribut ne doit pas contenir de caractère « espace », on peut utiliser les caractères « - » ou « _ »

Correspond

`<personne prenom = "jean"/>`

- Le **nom de l'élément** est l'élément auquel s'applique la définition de l'attribut
- Les **options** permettent de définir si l'attribut est obligatoire ou explicite
- **Défaut** correspond à la valeur par défaut attribuée à l'attribut

Les DTD et schémas XML – Les attributs

➤ Le *type* permet de définir des critères sur la valeur de l'attribut ou le type de données requis (caractères ou autres). Le *type* de l'attribut est à choisir parmi une liste prédéfinie :

- CDATA
- ENTITY ou ENTITIES
- NMTOKEN ou NMTOKENS
- NOTATION
- *Énumération*
- ID, IDREF ou IDREFS

Les DTD et schémas XML – Les attributs

Type	Utilisation	Exemple (DTD)	Exemple (XML)
CDATA	Seules les données de type caractères peuvent être employées dans l'attribut	<pre><!ELEMENT elt EMPTY> <!ATTLIST elt attribut CDATA "default" ></pre>	<pre><elt attribut= "valeur"/> <i>Par défaut :</i> <elt attribut= "default"/></pre>
ENTITY	La valeur de l'attribut doit faire référence à une entité binaire externe déclarée dans la DTD	<pre><!ELEMENT elt EMPTY> <!ENTITY animation SYSTEM ".../anim fla" NDATA flash > <!ATTLIST elt attribut ENTITY ></pre>	<pre><elt attribut= "animation"/> La valeur de l'attribut fait référence à l'entité binaire <i>animation</i></pre>
ENTITIES	Comme le type ENTITY mais autorise plusieurs valeurs séparées par des espaces		
NOTATION	La valeur de l'attribut doit faire référence à une notation déclarée quelque part dans la DTD (<!NOTATION...>)	<pre><!ELEMENT elt EMPTY> <!ATTLIST elt attribut NOTATION GIF> <!NOTATION GIF SYSTEM "/usr/bin/gifview.exe" ></pre>	<pre><elt attribut= "GIF"/> La valeur de l'attribut fait appel à la notation GIF</pre>

Les DTD et schémas XML – Les attributs

Type	Utilisation	Exemple (DTD)	Exemple (XML)
<p>Énumération</p> <p>Intérêt des attributs par rapport aux éléments</p>	<p>Une liste de valeurs entre parenthèses séparées par des barres verticales ()</p>	<pre><!ELEMENT elt EMPTY> <!ATTLIST elt attribut (v1 v2 v3) "v1"></pre>	<pre><elt attribut= "v1"/> Ou <elt attribut= "v2"/> Par défaut : <elt attribut= "v1"/></pre>
ID	<p>La valeur de l'attribut doit être un identificateur unique</p>		
IDREF	<p>La valeur doit faire référence à un identificateur ID déclaré quelque part dans le document</p>		
IDREFS	<p>Identique à IDREF mais autorise plusieurs valeurs</p>		

Les DTD et schémas XML – Les attributs

Type	Utilisation	Exemple (DTD)	Exemple (XML)
NMTOKEN	<p>La valeur de l'attribut est une combinaison quelconque de caractères qui peuvent être des lettres, des chiffres, des points, des barres obliques, des soulignés ou des deux-points.</p> <p>Il permet aussi d'affecter un nom symbolique à l'attribut tel qu'un format de date, un format de fichier ou encore une abréviation.</p>	<pre><!ELEMENT gestionnaire (répertoire)> <!ELEMENT répertoire (#PCDATA)> <!ATTLIST répertoire fichier NMTOKEN #REQUIRED></pre>	<pre><gestionnaire> <répertoire fichier= "index.htm" > Ce fichier représente la page d'accueil de ce répertoire </répertoire> </gestionnaire></pre>
NMTOKENS	<p>Identique à NMTOKEN mais autorise plusieurs valeurs séparées par des espaces</p>	<pre><!ELEMENT France (région)> <!ELEMENT région (#PCDATA)> <!ATTLIST région département NMTOKENS #REQUIRED></pre>	<pre><France> <région département= "14 50 61" > Basse-Normandie </région> </France></pre>

Les DTD et schémas XML – Les attributs

- Un attribut de type ID permet d'attribuer un identifiant à un élément. Cet élément peut être ensuite référencé dans une autre partie du même document en utilisant un attribut de type IDREF. Les identifiants permettent de lier différentes parties d'un document XML et réduire la saisie de données dans un document
- L'exemple ci-dessous permet de définir des personnes et leur lien de parenté en utilisant les types ID et IDREF

```
<!ELEMENT personne (#PCDATA)>  
<!ELEMENT parent (fils*)>  
<!ELEMENT fils EMPTY>  
<!ATTLIST personne identifiant ID #REQUIRED>  
<!ATTLIST parent identifiant IDREF #REQUIRED>  
<!ATTLIST fils identifiant IDREF #REQUIRED>
```

Définition de l'identifiant sur la
personne

Référence à l'identifiant
pour le parent ou le fils

Les DTD et schémas XML – Les attributs

- Le document XML respectant la DTD, contient la définition de deux personnes (Adam et Caïn) à qui on attribue un identifiant (respectivement « A » et « B »), puis on précise que A est parent de B

```
<personne identifiant="A">Adam</personne>
<personne identifiant="B">Caïn</personne>
<parent identifiant="A">
  <fils identifiant="B"/>
</parent>
```

Mise en correspondance
des données et de
l'identifiant :

ID « A » => « Adam »

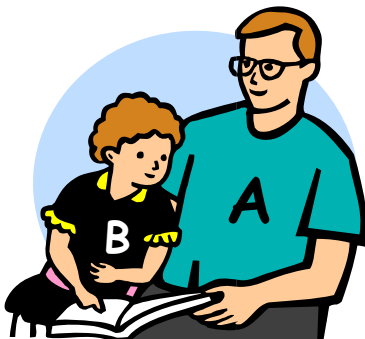
ID « B » => « Caïn »

On fait référence aux données à l'aide des identifiants :

`<parent identifiant="A">` est équivalent à
`<parent>Adam</Parent>`

`<fils identifiant="B">` est équivalent à `<fils>Caïn</fils>`

Donc les identifiants réduisent la saisie multiple de
données dans le document XML



Les DTD et schémas XML – Les attributs

➤ Il est à noter des contraintes sur l'utilisation des identifiants (ID et IDREF)

Contrainte	Exemple (DTD)	Exemple (XML)
<p>Les ID sont uniques pour l'ensemble du document XML</p>	<pre><!ELEMENT personne (fils*)> <!ELEMENT fils EMPTY> <!ATTLIST personne identifiant ID #REQUIRED> <!ATTLIST fils identifiant2 ID #REQUIRED></pre>	<p><i>Cette notation n'est pas valide :</i></p> <pre><personne identifiant="n01"> <fils identifiant2="n01"/> <fils identifiant2="n02"/> </personne></pre> <p><i>Cette notation est valide :</i></p> <pre><personne identifiant="n01"> <fils identifiant2="n02"/> <fils identifiant2="n03"/> </personne></pre>
<p>La valeur de l'ID doit toujours commencer par une lettre</p>	<pre><!ELEMENT fils EMPTY> <!ATTLIST fils identifiant2 ID #REQUIRED></pre>	<pre><fils identifiant2="01"/> invalide <fils identifiant2="n01"/> est valide</pre>

Les DTD et schémas XML – Les attributs

Contrainte	Exemple (DTD)	Exemple (XML)
<p>On ne peut utiliser l'option <i>#FIXED</i> avec un attribut ID</p>	<pre><!ELEMENT personne (fils*)> <!ELEMENT fils EMPTY> <!ATTLIST personne identifiant ID #REQUIRED> <!ATTLIST fils identifiant2 ID #FIXED "n01"></pre>	<pre><fils identifiant2="n02"/>"n01"/> <p>On comprend par ces exemples qu'on ne peut utiliser l'option <i>#FIXED</i> avec un attribut ID</p> </pre>
<p>Un élément ne peut avoir plus d'un ID</p>	<pre><!ELEMENT fils EMPTY> <!ATTLIST fils identifiant2 ID #REQUIRED></pre>	<pre><fils identifiant2="n01" "n02"/> </pre>

Les DTD et schémas XML – Les attributs

- A la déclaration de l'attribut on peut ajouter des « options » pour indiquer le caractère obligatoire ou optionnel de l'attribut. On peut choisir parmi une liste d'options :
- **#REQUIRED**
 - **#IMPLIED**
 - **#FIXED** *valeur*

Les DTD et schémas XML – Les DTD

Type	Utilisation	Exemple (DTD)	Exemple (XML)
#REQUIRED	L'attribut doit obligatoirement avoir une valeur explicite	<pre><!ELEMENT auteur EMPTY> <!ATTLIST auteur Nom CDATA #REQUIRED></pre>	<p><auteur/> <i>invalide</i></p> <p><auteur Nom="Dupont"/> <i>est valide</i></p>
#IMPLIED	L'attribut est optionnel	<pre><!ELEMENT auteur EMPTY> <!ATTLIST auteur Nom CDATA #IMPLIED></pre>	<p><auteur/> <i>est valide</i></p> <p><auteur Nom="Dupont"/> <i>est valide</i></p>
#FIXED valeur	L'attribut doit avoir la valeur indiquée	<pre><!ELEMENT auteur EMPTY> <!ATTLIST auteur Editeur CDATA #FIXED "Eyrolles"></pre>	<p><auteur Editeur="MicroApp"/> <i>invalide</i></p> <p><auteur Editeur="Eyrolles"/> <i>est valide</i></p>

Les DTD et schémas XML – Les DTD

- Plusieurs attributs concernant un élément peuvent être déclarés au sein de la même balise **ATTLIST** :

<!ATTLIST

[nom de l'élément] [nom de l'attribut1] [type] [options] [défaut]

[nom de l'élément] [nom de l'attribut2] [type] [options] [défaut]

...>

Le rappel du nom de l'élément n'est pas obligatoire

- Par exemple :

<!ELEMENT SPECIFICATION EMPTY>

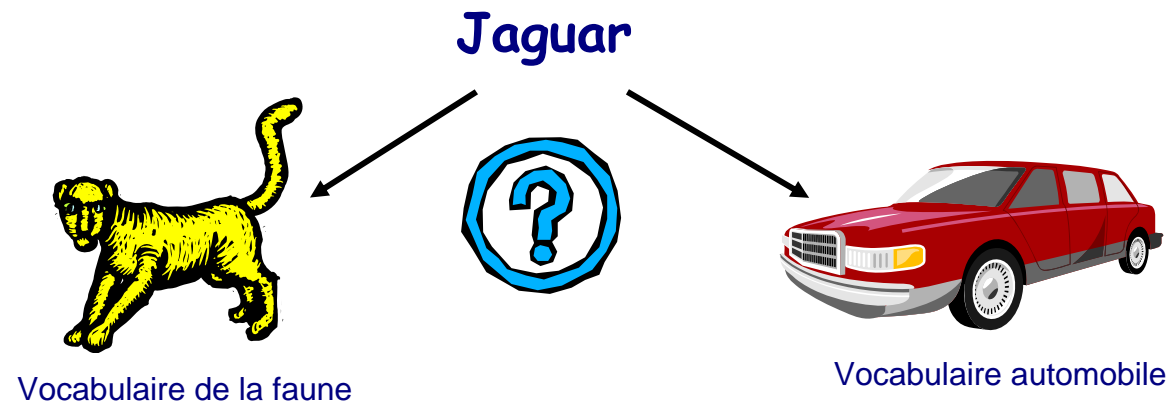
<!ATTLIST SPECIFICATION PROVENANCE CDATA #REQUIRED COULEUR CDATA #REQUIRED>

Correspond

<SPECIFICATION PROVENANCE="Bordeaux" COULEUR="Rouge"/>

Les DTD et schémas XML – Les namespaces

- Les « **espaces de nommage** » (ou namespaces) est une recommandation du W3C dont l'objectif est :
- distinguer les éléments et les attributs provenant de différents vocabulaires (exemple jaguar : faune ou automobile) et lever les ambiguïtés éventuelles sur l'intitulé de la balise
 - Regrouper les éléments et les attributs corrélés de manière à ce que la réutilisation soit plus facile



Les DTD et schémas XML – Les namespaces

- Un document contient les informations générales concernant l'entreprise (nom, adresse, raison sociale...) :

La balise « nom » peut contenir 2 types d'informations : nom de l'entreprise ou nom de l'employé. Si l'on veut créer un seul document, comment distinguer les deux ?

```
<entreprise>  
  <nom> Bizib inc  
  </nom>  
  <adresse>  
    ...  
  </adresse>  
</entreprise>
```



- Un autre document contient en revanche les informations concernant les employés (nom, prénom, fonction...) :

```
<personnes>  
  <personne>  
    <nom>  
      <nomDeFamille>Lafargue</nomDeFamille>  
      <prenom>Paul</prenom>  
    </nom>  
    <fonction>PDG</fonction>  
    <telephone>0102030405</telephone>  
  </personne>  
</personnes>
```



Les DTD et schémas XML – Les namespaces

- A l'aide des espaces de nommage on va pouvoir différencier les balises `<nom>` en utilisant la notation `<entreprise:nom>` et `<personne:nom>`

- L'espace de nommage « personne » pourra être déclaré de la façon suivante :

`xmlns:personne="http://www.personne.org"`

de même pour l'entreprise : `xmlns:entreprise="http://www.entreprise.org"`

L'URI de l'espace de nommage peut être fictive car elle n'est pas vérifiée, toutefois elle pointe généralement sur la grammaire de l'espace de nommage (DTD ou schéma XML)

- Ensuite on pourra préfixer l'élément « nom » soit par « personne » ou « entreprise », ce qui signifie la balise « nom » appartient à l'espace de nommage

Les DTD et schémas XML – Les namespaces

➤ Le document XML pourra donc se présenter comme suit :

Grammaire pour l'entreprise

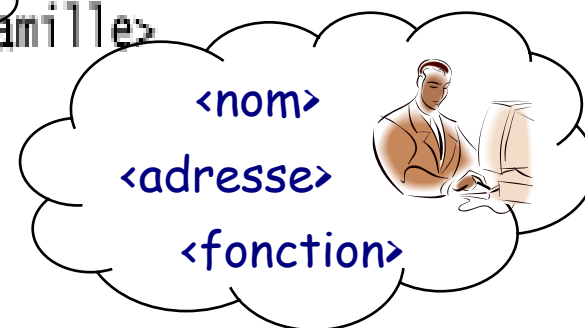


Déclaration des espaces de nom dans
l'élément racine

```
<organisation  
  xmlns:entreprise="http://www.entreprise.org"  
  xmlns:personne="http://www.personne.org">  
  <entreprise:nom>Bizib inc</entreprise:nom>  
  <personne:nom>  
    <personne:nomDeFamille>Lafargue</personne:nomDeFamille>  
    <personne:prenom>Paul</personne:prenom>  
  </personne:nom>  
  <personne:fonction>PDG</personne:fonction>  
</organisation>
```

Références aux espaces de nom

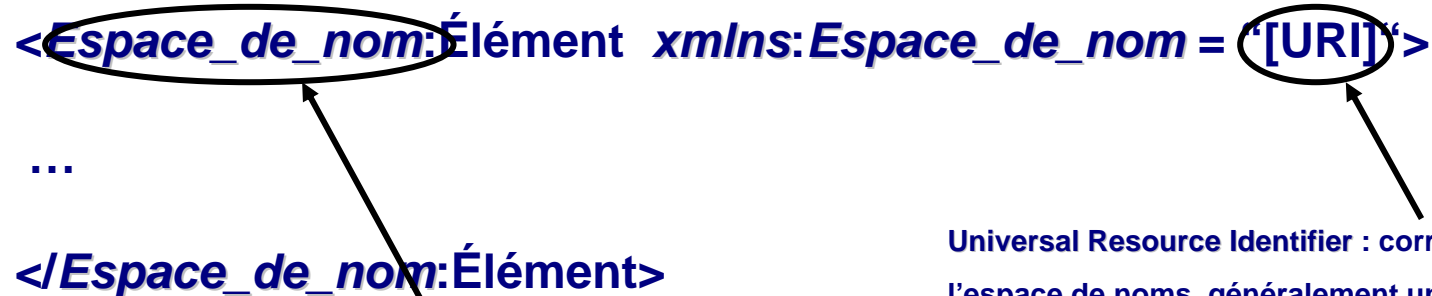
Grammaire pour l'employé



Les DTD et schémas XML – Les namespaces

- Les espaces de nommage peuvent être déclarés dans le document XML à l'aide de l'attribut *xmlns*

```
<Espace_de_nom:Élément xmlns:Espace_de_nom = "[URI]">  
...  
</Espace_de_nom:Élément>
```



Universal Resource Identifier : correspond à l'emplacement de l'espace de noms, généralement un fichier DTD ou schéma XML. Toutefois, l'emplacement de l'espace de nom peut être fictif car il n'est pas vérifié

Cette notation dans la balise de déclaration est optionnelle, on aurait pu aussi indiquer :

```
<Élément xmlns:espace_de_nom="[URI]"/>
```

- La **déclaration de l'espace de nom** peut figurer dans tout élément du document XML, ou pour plus de clarté et de commodité dans l'élément racine

Les DTD et schémas XML – Les namespaces

- Les éléments appartenant à un espace de nom se distinguent des autres éléments par l'ajout d'un préfixe symbolisant cette singularité. Les préfixes se placent au sein de la balise XML avant le nom de l'élément et séparés par un « : »

<Espace_de_nom:Élément > [donnée] </Espace_de_nom:Élément>

- Le nom du préfixe peut être constitué de lettres, caractères accentués, tirets (« _ ») ou de chiffres

Les DTD et schémas XML – Les namespaces

- Un espace de nommage peut aussi figurer dans la DTD et en particulier dans la définition de l'attribut :

<!ELEMENT Espace_de_nom:[nom de l'élément] ([contenu de l'élément ou type])>

<!ATTLIST Espace_de_nom:[nom de l'élément] xmlns:Espace_de_nom CDATA [options] "[URI]">

Emplacement de l'espace de nommage doit apparaître en valeur par défaut

- Soit par exemple :

<!ELEMENT entreprise:nom #PCDATA>

<!ATTLIST entreprise:nom xmlns:entreprise CDATA #IMPLIED "http://www.entreprise.org">

Correspond

<entreprise:nom xmlns:entreprise= "http://www.entreprise.org">

....

</entreprise:nom>

<entreprise:nom> Dupont et Cie </entreprise:nom>

Atelier

Création d'une DTD



Les DTD et schémas XML – Les schémas XML

- **Les DTD répondent en grande partie au besoin de définir des contraintes et des règles de validation sur les documents XML. Toutefois, ils présentent des limitations qui les rendent obsolètes quand il s'agit de détailler plus finement les règles de validation :**
 - **La syntaxe des DTD n'est pas en XML : Le fait d'écrire un document XML en utilisant une syntaxe différente est quelque peu contradictoire**
 - **Le typage de données pauvre : la DTD présente peu de possibilité de typer des données (CDATA, NMTOKEN et #PCDATA)**
 - **La gestion des espaces de nom est limitée**
 - **On ne peut définir précisément le nombre de sous éléments (0-5 ou 1-15)**

Les DTD et schémas XML – Les schémas XML

- A partir de février 2000 le W3C a proposé une nouvelle norme, le *schéma XML* ou *XSD (eXtensible Schema Definition)* dont l'objectif est de remplacer à terme la DTD. Les points forts des schémas XML sont :
 - **Extensibles** : on peut créer de nouveaux types de données ou enrichir les types existants
 - **Écrits en XML**
 - **Capables de définir des éléments de même nom mais de contenu différent**
 - **Plus riches** : on peut détailler de manière précise le nombre d'occurrence de sous éléments (1-5 ou 0-21)
 - **Capables de gérer de nouvelles notions telles que l'héritage des éléments ou des attributs**

Les DTD et schémas XML – Les schémas XML

- Prenons le document XML ci-dessous :

```
<entree>
  <nom>Harry Cover</nom>
  <telephone>0102030405</telephone>
</entree>
```

Le langage XSD s'appuie sur
l'espace de nom « xsd »

- Le schéma XML nécessaire à sa validation se présente comme suit :

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:element name="entree">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element
          name="nom"
          type="xsd:string"
          minOccurs="1"
          maxOccurs="1"/>
        <xsd:element
          name="telephone"
          type="xsd:decimal"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Concept de « séquence » pour
créer un nouveau type pris de
l'ASN1

On note que XSD est plus
« verbeux » que la DTD, en
revanche il permet d'être plus
précis

Les DTD et schémas XML – Les schémas XML

- Pour se référer à un schéma XML, il faut le préciser dans l'entête du document

XML :

```
<entree xmlns="http://www.annuaire.org"
        xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
        xsi:schemaLocation="http://www.annuaire.org/entree.xsd">
  <nom>Harry Cover</nom>
  <telephone>0102030405</telephone>
</entree>
```

- `xmlns="http://www.annuaire.org"` : est l'espace de nom utilisé par défaut dans le document
- `xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"` : indique que le document est une instance d'un schéma XML (notation obligatoire)
- `xsi:schemaLocation="http://www.annuaire.org/entree.xsd"` : indique le nom du schéma XML (.xsd) qui valide le document (*schemaLocation* contient la paire de valeurs : espace de noms et fichier de définitions du document)

Les DTD et schémas XML – Les schémas XML

- Pour déclarer un schéma XML on utilise l'élément `<xsd:schema>` associé à des paramètres :

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
  targetNamespace="http://www.annuaire.org"
  xmlns="http://www.annuaire.org"
  elementFormDefault="qualified"/>
```

- `xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"` : les types utilisés pour construire le schéma XML viennent de cet espace de nom (notation obligatoire)
- `targetNamespace="http://www.annuaire.org"` : les éléments définis dans le schéma XML sont contenus dans cet espace de nom cible
- `xmlns="http://www.annuaire.org"` : correspond à l'espace de nom par défaut
- `elementFormDefault="qualified"` : tous les documents qui se réfèrent à ce schéma XML doivent répondre à l'espace de nom défini

Les DTD et schémas XML – Les schémas XML

- Pour définir un élément on utilise la balise ***<xsd:element>*** :

```
<xsd:element name= "[nom élément]" type= "[xsd:type]" minoccurs= "[0 à n]" maxoccurs= "[0 à n]">
```

...

```
</xsd:element>
```

- Les attributs optionnels `minoccurs` et `maxoccurs` permettent de préciser le nombre d'occurrences min et max de sous éléments. Pour définir un nombre indéterminé d'occurrences il faut utiliser le mot clé « *unbounded* »

Les DTD et schémas XML – Les schémas XML

- On peut établir une correspondance entre les notations DTD et XSD :

Notation XSD

```
<xsd:element name="élément">  
  <xsd:complexType>  
    <xsd:sequence>  
      <xsd:element name="s1">  
        minoccurs = "0"  
        maxoccurs = "unbounded" />  
    </xsd:sequence>  
  </xsd:complexType>  
</xsd:element>
```

Notation DTD

```
<!ELEMENT élément (s1*)>
```

```
...  
<xsd:element name="s1">  
  minoccurs = "0"  
  maxoccurs = "1" />  
...
```

```
<!ELEMENT élément (s1?)>
```

Les DTD et schémas XML – Les schémas XML

- On peut établir une correspondance entre les notations DTD et XSD :

Notation XSD	Notation DTD
... <xsd:sequence> <xsd:any/> </xsd:sequence> ...	<!ELEMENT élément ANY >
... <xsd:choice> <xsd:element name="s1" /> <xsd:element name="s2" /> </xsd:choice> ...	<!ELEMENT élément (s1 s2)>

Les DTD et schémas XML – Les schémas XML

- Pour définir un attribut on utilise la balise **<xsd:attribute>** :

```
<xsd:attribute name= "[nom attribut]" type= "[xsd:type]" use= "[options]"  
value= "[valeur par défaut]">
```

...

```
</xsd:attribute>
```

- L'attribut **use** permet de définir des contraintes sur la valeur de l'attribut, identiques à la DTD soit : *required* (obligatoire), *IMPLIED* (optionnel) ou *fixed* (valeur fixe)
- Et tout comme la DTD il est possible d'indiquer une valeur par défaut (*value*)

Les DTD et schémas XML – Les schémas XML

- Ci-dessous un exemple de déclaration d'un attribut :

```
<xsd:element name="element">
```

```
  <xsd:complexType>
```

```
    <xsd:attribute name="att"
```

```
      type="xsd:string"
```

```
      use="implied"
```

```
      value="a"/>
```

```
    <xsd:attribute name="at2"
```

```
      type="xsd:boolean"
```

```
      use="required"
```

```
      value="true"/>
```

```
  </xsd:complexType>
```

```
</xsd:element>
```

Déclaration de l'attribut
effectuée à l'intérieur des
balises <xsd:element>

Définition de l'option
(« implied » soit optionnel)
et de la valeur par défaut
(« true »)

Les DTD et schémas XML – Les schémas XML

- La norme XSD définit près de 40 types de données : string, integer, decimal, float, date, year, boolean, CDATA...
- Toutefois, il est possible de créer un nouveau type de données en utilisant la balise `<xsd:complexType>`. Il est à noter que la définition d'élément et de sous éléments implique la création d'un nouveau type de données
- Par exemple pour créer un type « typeEntrée » d'annuaire :

```
<entree>  
  <nom>string</nom>  
  <telephone>decimal</telephone>  
</entree>
```

On utilise la syntaxe

```
<xsd:complexType name="typeEntree">  
  <xsd:sequence>  
    <xsd:element name="nom" type="xsd:string"/>  
    <xsd:element name="telephone" type="xsd:decimal"/>  
  </xsd:sequence>  
</xsd:complexType>
```

Les DTD et schémas XML – Les schémas XML

- Puis on associe le nouveau type de données à l'élément entrée d'annuaire

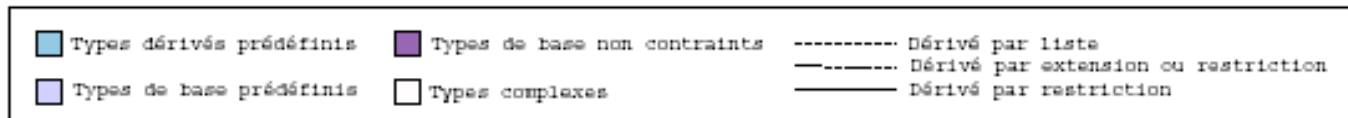
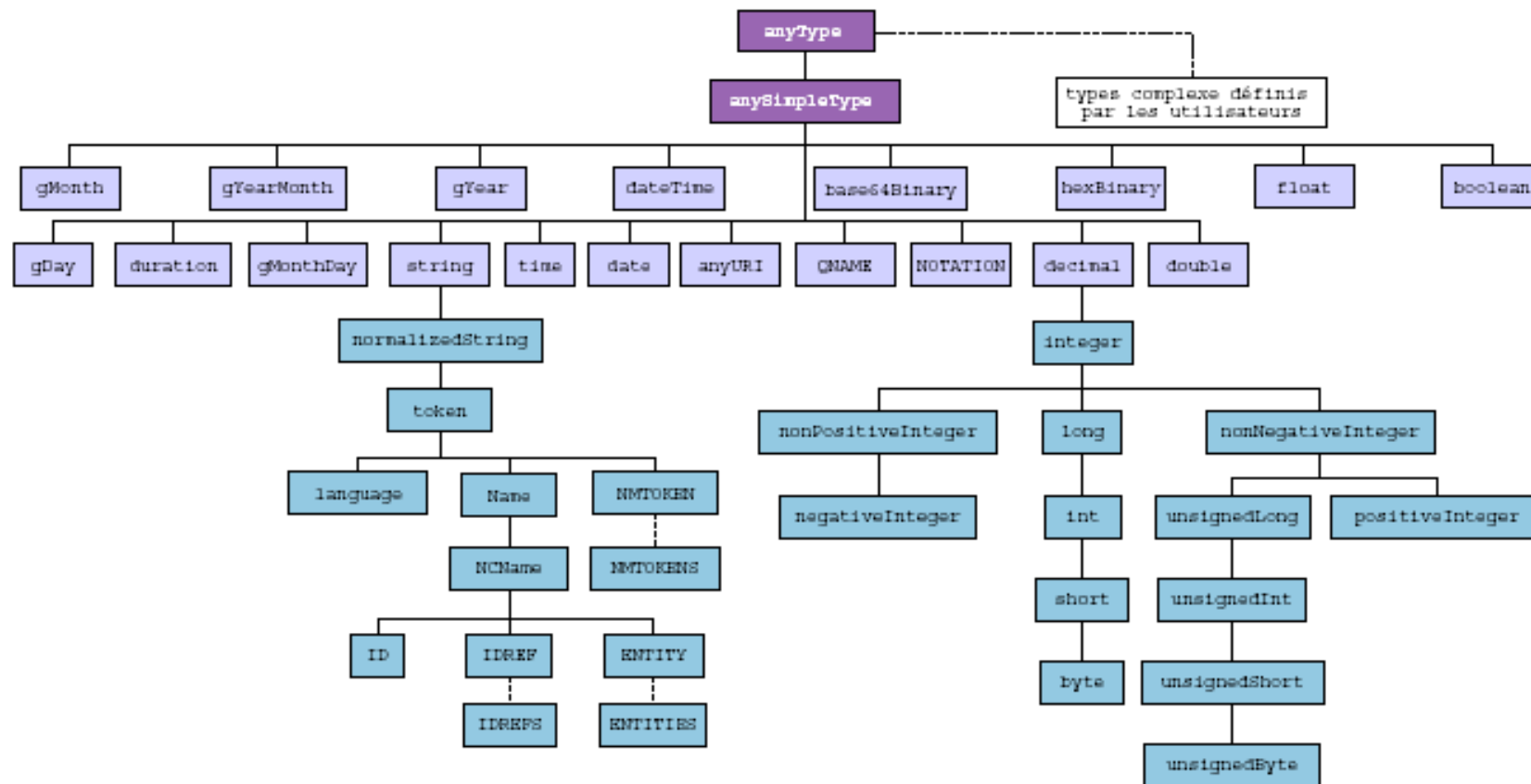
```
<xsd:element name="entree" type="typeEntree">
```

- On peut aussi étendre un type de donnée existant en ajoutant un élément, c'est un *type dérivé*. Par exemple, on va étendre le type « typeEntrée » et créer le nouveau type « entréeAvecadresse ». On utilise les balises **<xsd:extension>** et **<xsd:complexContent>**

```
<xsd:complexType name="entreeAvecAdresse">  
  <xsd:complexContent>  
    <xsd:extension base="entree" >  
      <xsd:sequence>  
        <xsd:element  
          name="adresse" type="xsd:string"/>  
      </xsd:sequence>  
    </xsd:extension>  
  </xsd:complexContent>  
</xsd:complexType>
```

Les DTD et schémas XML – Les schémas XML

- Le tableau ci-dessous présente l'ensemble des types de données définis par XSD :



Atelier

Transformation d'une DTD en schéma XML



Questions ?



XML

Les bases en pratique

Support de cours réalisé par Joël FARVAULT (CNRS/DSI) et David ROUSSE (CNRS/DSI), relu par René PELFRESNE (CNRS/DSI)



Ce(tte) oeuvre est mise à disposition selon les termes de la Licence Creative Commons Paternité - Pas d'Utilisation Commerciale - Pas de Modification 2.0 France.

Avril 2003

Direction des systèmes d'information

- Tous les exemples présentés peuvent être obtenus par mail : david.rousse@dsi.cnrs.fr
- L'ensemble des exemples présentés dans les parties Transformations de documents XML, Liaisons de documents et Manipulations de documents XML a été testé sur la configuration suivante :
 - Windows XP SP1
 - Sun JDK 1.4.0_01 (installé dans C:\j2sdk1.4.0_01)
 - Apache Xerces 2.3.0 (installé dans C:\j2sdk1.4.0_01\xerces-2_3_0)
 - Apache Xalan 2.4.1 (installé dans C:\j2sdk1.4.0_01\xalan-j_2_4_1 mais copie xalan.jar de dans C:\Program Files\Java\j2re1.4.0_01\lib\endorsed)
 - Apache FOP 0.20.4 (installé dans C:\j2sdk1.4.0_01\fop-0.20.4)
 - PATH=%PATH%;C:\j2sdk1.4.0_01\bin
 - CLASSPATH=C:\j2sdk1.4.0_01\xerces-2_3_0\xercesSamples.jar;C:\j2sdk1.4.0_01\xerces-2_3_0\xercesImpl.jar;C:\j2sdk1.4.0_01\xerces-2_3_0\xml-apis.jar;C:\j2sdk1.4.0_01\xerces-2_3_0\xmlParserAPIs.jar;C:\j2sdk1.4.0_01\fop-0.20.4\build\fop.jar;C:\j2sdk1.4.0_01\fop-0.20.4\lib\batik.jar;C:\j2sdk1.4.0_01\fop-0.20.4\lib\avalon-framework-cvs-20020315.jar;C:\j2sdk1.4.0_01\fop-0.20.4\lib\logkit.jar;C:\j2sdk1.4.0_01\fop-0.20.4\lib\jimi-1.0.jar;
 - JAVA_HOME=C:\j2sdk1.4.0_01
- Il est possible de vérifier votre configuration ainsi :
 - lancer pour tester Xalan : java org.apache.xalan.xslt.EnvironmentCheck
 - lancer pour tester Xalan (Process) : java org.apache.xalan.xslt.Process -v
 - lancer pour tester FOP : java org.apache.fop.apps.Fop

Introduction

Transformations de documents XML

Principes généraux

CSS

XSLT

XSL-FO

XPath

Liaisons de documents en XML

Principes généraux

XLink

XPointer

Manipulations de documents XML

Principes généraux

SAX

DOM

Conclusion

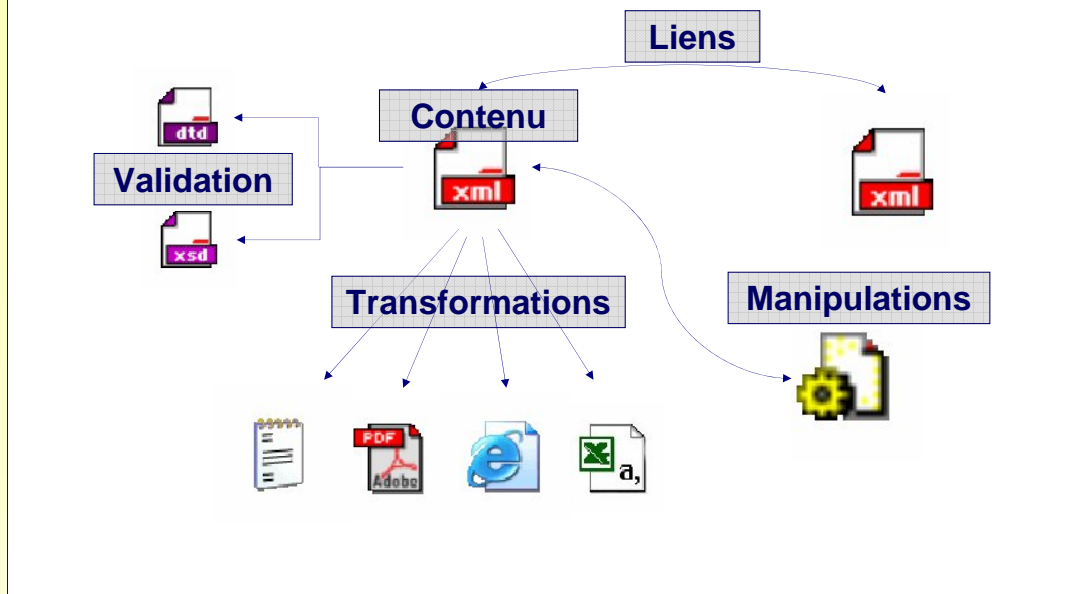
Références

- **2 ateliers seront réalisés, pour la partie Transformations.**

Introduction

Avril 2003

Direction des systèmes d'information



- La 1° partie était consacrée principalement au contenu (document XML) et à la structure du contenu (DTD et schémas XML).
- La 2° partie sera dédiée aux transformations (CSS, XSLT, XSL-FO, XPath), aux liaisons de documents XML (XLink, XPointer) et aux manipulations de documents XML (SAX, DOM).
- Quelques outils pour éditer du XML, autres que les outils du groupe Apache :
 - Cooktop 2.5 (utilise MSXML 4.1).
 - Amaya 7.1 du W3C (implante XLink notamment).
 - GenDoc 1.0 (de SourceForge, en JAVA).
 - eXchanger v0.09 (xngr.org, en JAVA).
 - XMLSpy 5.0 Entreprise (payant de la société Altova, meilleur outil de la liste présentée ici à mon sens).

-
- Le document XML contient les données (notion de document bien formé)**
 - Les DTD ou les schémas XML permettent de décrire des contraintes sur ce que peut contenir un document XML (notion de document valide)**
 - Les langages de transformations ont pour objet de transformer un document XML en « un autre document » (présenté à l'utilisateur ou destiné à des applications)**
 - La notion de liens représente un exemple d'application XML : comment exprimer des liens avec une syntaxe XML ?**
 - Les manipulations de documents XML depuis un langage de programmation passent par l'utilisation d'APIs normalisées**

Transformations de documents XML

Avril 2003

Direction des systèmes d'information

Introduction

Transformations de documents XML

Principes généraux

CSS

XSLT

XSL-FO

XPath

Liaisons de documents en XML

Principes généraux

XLink

XPointer

Manipulations de documents XML

Principes généraux

SAX

DOM

Conclusion

Références

- Le document XML contient les données et leur sémantique
- Les règles de présentation et/ou de transformation sont stockées dans des fichiers annexes
- Les règles de présentation et/ou de transformation sont exprimées avec :
 - CSS (Cascading StyleSheets)
 - XSL (eXtensible Stylesheet Language)
 - XSLT (XSL Transformations)
 - XSL-FO (XSL-Formatting Objects)
- Les règles de présentation et/ou de transformation sont appliquées au contenu du document XML
- Un langage de requêtes, XPath, permet de sélectionner des fragments d'un document XML

- Dans certains ouvrages, XSL est synonyme de XSL-FO.

Introduction

Transformations de documents XML

Principes généraux

CSS

XSLT

XSL-FO

XPath

Liaisons de documents en XML

Principes généraux

XLink

XPointer

Manipulations de documents XML

Principes généraux

SAX

DOM

Conclusion

Références

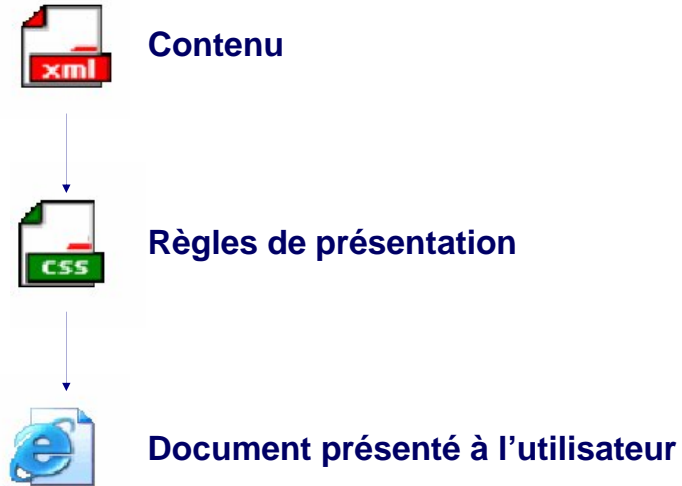
- Langage non XML spécifié par le W3C
- Permet de décrire la manière de présenter le contenu d'un document XML
- Exemple :

```
/* Propriétés par défaut pour l'ensemble du document */
recette {
    font-family: "New York", "Times New Roman", serif;
    font-size: 12pt
}

/* Titre principal */
plat {
    display: block;
    font-family: Helvetica, Arial, sans-serif;
    font-size: 20pt;
    font-weight: bold;
    text-align: center
}
...
```

- Spécifications des feuilles de modèle version 2 disponible à l'adresse <http://www.w3.org/Style/CSS/>

- Dédié à la présentation de documents XML en HTML



- En source, on peut avoir du HTML ou du XML.
- En cible, on a le plus souvent du HTML
- Plus généralement, CSS2 permet de générer du Web écrit, du Web imprimé et du Web oral.

CSS – Avantages et inconvénients

Avantages

- Langage simple
- Langage implanté dans les navigateurs actuels

Inconvénients

- Traite les éléments dans l'ordre dans lequel ils apparaissent dans le document source
- Restreint à la couche présentation (les transformations telles que le tri, le calcul ne sont pas exprimables en CSS)

- Netscape version 7.0 et Internet Explorer version 5.0 implantent tout deux CSS 1 et 2 (pour un liste plus complète, voir <http://www.w3.org/Style/CSS/#browsers>).

☐ 3 niveaux disponibles

- **Niveau 1 : destiné à HTML**
- **Niveau 2 : destiné à HTML et XML**
- **Niveau 3 :**
 - **apporte de la modularité à la spécification pour permettre aux applications d'en implanter seulement une partie**
 - Exemple : application implantant le module dédié à la synthèse vocale et n'implantant pas le formatage visuel
 - **Spécifie de nouvelles fonctions**
 - Exemple : mise en page multi colonnes

- **La version 2 est implantée dans les navigateurs récents.**
- **Actuellement, le niveau 3 est au stade de Working Draft.**

□ Document XML source

Instruction de traitement permettant d'associer une feuille CSS au document XML

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<?xml-stylesheet type="text/css" href="presentation_recette.css"?>
<recette source="Gigi">
  <plat>Crêpe "Gigi"</plat>
  <ingrédients>
    <ingrédient>
      <quantite>1</quantite>
      <produit>sarrazin</produit>
    </ingrédient>
    <ingrédient>
      <quantity>1</quantity>
      <component>crêpe</component>
    </ingrédient>
    <ingrédient>
      <quantity>Sucre</quantity>
      <component>10g</component>
    </ingrédient>
  </ingrédients>
  <etapes>
    <etape>Mettre une fine couche de sarrazin</etape>
    <etape>Saisir dessus-dessous</etape>
  </etapes>
  <alternative>Plat alternatif : crêpe au sucre avec une bière</alternative>
</recette>
```

- Il est possible de mettre en place plusieurs instructions de traitements dans le prologue pour prendre en compte plusieurs périphériques de sortie :

```
<?xml-stylesheet type="text/css" media="screen" title="Pour le Web" href="presentation_recette.css"?>
<?xml-stylesheet type="text/css" media="print" title="Pour l'impression" href="presentation_recette2.css"?>
```


CSS – Exemple (2/3)

❑ Feuille CSS

```
/* Propriétés par défaut pour l'ensemble du document */
recette {
    font-family: "New York", "Times New Roman", serif;
    font-size: 12pt
}
/* Titre principal */
plat {
    display: block;
    font-family: Helvetica, Arial, sans-serif;
    font-size: 20pt;
    font-weight: bold;
    text-align: center
}
/* Liste à puces */
ingredient, etape {
    display: list-item;
    margin-left: 40pt
}
/* Création de 2 paragraphes */
etapes, alternative {
    display: block;
    margin-top: 12pt;
}
```

Commentaire

Élément auquel appliquer une règle de styles de présentation

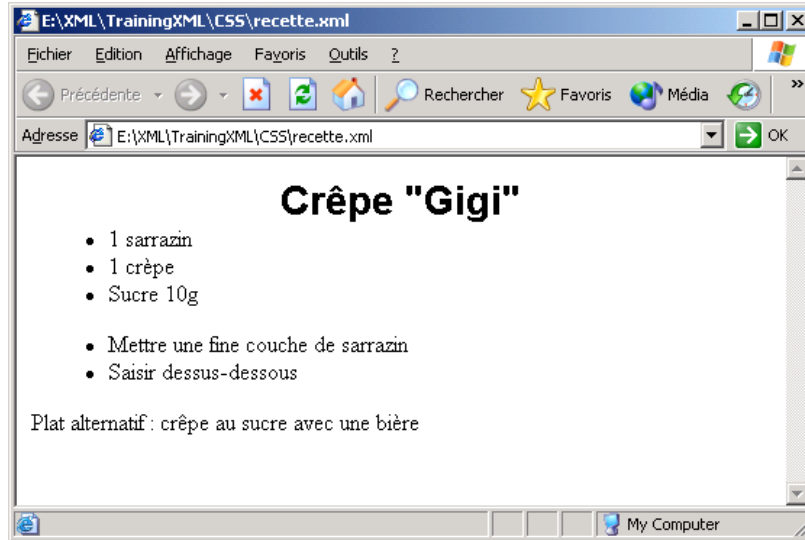
Chaque règle nomme le(s) élément(s) qu'elle formate (ici plat) et contient entre {...} les propriétés de style à appliquer

Nom de la propriété

Valeur de la propriété

- Exemple :
 - 1 – élément plat
 - 2 – règle à appliquer à plat
 - 3 – nom de propriété
 - 4 – valeur de propriété
 - 5 – commentaire
-
- 4 règles de style sont écrites ici.
 - En CSS2, une dizaine de propriétés sont disponibles.
 - Il existe des propriétés par défaut.

□ Feuille HTML résultat



□ Au lieu de désigner les éléments du document XML par leurs noms, il est possible d'utiliser des sélecteurs

- Le sélecteur universel via *
 - permet d'appliquer une règle à tout élément avec lequel aucune règle n'entre en conflit
 - Exemple : * { font-size: 14pt; }
- Les sélecteurs descendants, enfants et frères
 - permet d'appliquer une règle à tous les descendants d'éléments
 - Exemple : recette ingredients { font-size: 14pt; }
 - permet d'appliquer une règle à tous les enfants d'un élément
 - Exemple : ingredients { font-size: 18pt; }
ingredients > ingredient { font-size: inherit; }
 - permet d'appliquer une règle à tous les frères d'un élément
 - Exemple : etapes + alternative { font-size: 18pt; }
- Les sélecteurs d'attributs via [...]
 - permet de sélectionner des éléments ayant des attributs
 - Exemple : recette[source="Gigi"] { font-size: 18pt; }

- **recette ingredients { font-size: 14pt; } /* tout fils de *recette* appelé *ingredients* et tous descendants de cet *ingredients* */**
- **Tout *ingredient* hérite des propriétés de son parent *ingredients* (Remarque : utilisation du mot inherit)**
 - ingredients { font-size: 18pt; }
 - ingredients > ingredient { font-size: inherit; }
- **etapes + alternative { font-size: 18pt; } /* ! ne fonctionne pas ! */**
- **recette[source="Gigi"] { font-size: 18pt; } /* élément recette ayant un attribut source valant Gigi */**
- **Il n'existe pas de sélections pour atteindre le contenu d'un attribut d'un élément.**

□ Au lieu de désigner les éléments du document XML par leurs noms, il est possible d'utiliser des sélecteurs

- Les sélecteurs de pseudo-classes
 - permet de sélectionner les éléments selon un condition n'impliquant pas leur nom
 - Exemple : `*:hover { color: green; }`
- Les sélecteurs de pseudo-éléments
 - permet d'appliquer une règle à autre chose qu'à des éléments
 - Exemple : `produit:first-letter { font-weight: bold; }`

- Il existe 7 sélecteurs de pseudo-classes
- `*:hover { color: green; } /* affiche l'élément en vert quand la souris est dessus */`
- Il existe 4 sélecteurs de pseudo-éléments
- `produit:first-letter { font-weight: bold; } /* affiche la 1o lettre de produit en gras */`

□ Les propriétés importantes de CSS sont les suivantes

- **display** permet de présenter les éléments en blocs, listes et tables
 - Exemple :

```
etape {  
    display: list-item;  
    list-style-type: decimal }
```
- CSS fournit également des propriétés relatives à la longueur, comme *border-width*, *font-size*, *line-height*, *margin-left*, ...
- Les fontes sont également manipulables via les propriétés *font-family*, *font-style*, *font-size*, *font-weight*, ...
- Le formatage du texte se réalise via *text-indent*, *text-align*, *text-decoration*, *text-transform*, ...
 - Exemple :

```
etape { text-transform: uppercase; }
```
- Les couleurs des éléments sont paramétrées via *color*, *background-color*, *border-color*

- **Exemples de propriétés :**
- Mise en page avec display
- Fonte
- Type de fonte
- Taille de fonte
- Couleur de fonte
- Etc ...

CSS – Conclusion

- CSS est destinée à la présentation de documents HTML et XML
- CSS est un langage simple
- CSS est implanté dans les navigateurs Web actuels
- Les transformations de documents XML doivent se faire avec un autre langage, XSLT

- **Durée prévue [Début – CSS] : 10 minutes.**
- **Atelier de 10 minutes.**

Introduction

Transformations de documents XML

Principes généraux

CSS

XSLT

XSL-FO

XPath

Liaisons de documents en XML

Principes généraux

XLink

XPointer

Manipulations de documents XML

Principes généraux

SAX

DOM

Conclusion

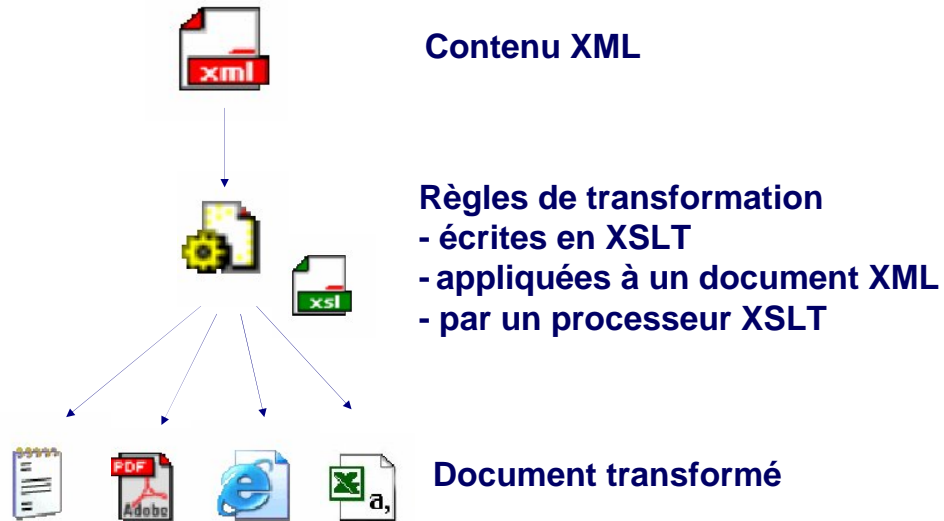
Références

XSLT – Définition ^(1/2)

- Langage XML spécifié par le W3C
- Permet de décrire des transformations à appliquer à un document XML
- Les règles de transformations sont écrites dans un document XSLT, encore appelé feuille de style XSLT
- Un processeur XSLT implante le langage XSLT et permet d'appliquer des règles de transformations XSLT sur un document XML source
 - Quelques processeurs XSLT :
 - Apache Xalan, xml.apache.org
 - Saxon de Michael Kay, saxon.sourceforge.net
 - XT de James Clark, www.jclark.com/xml
 - L'offre d'Oracle, otn.oracle.com/tech/xml

- Un document XSLT est un document XML bien formé.
- Spécifications de la version 1.0 disponible à l'adresse <http://www.w3.org/TR/xslt>

- Dédié à la transformations de documents XML



- On a en fait :
- CSS pour la présentation du HTML.
- XSLT pour la présentation/transformation du XML.
- DSSSL pour la présentation, du SGML.
- Remarque : le processeur XSLT Apache Xalan implante les spécifications de XSLT, XPath, SAX et DOM (via Xerces) et TraX (Transformation API for XML, TraX étant un sous-ensemble des spécifications JAXP de Sun).

Avantages

- Langage XML
- Langage puissant
- Langage de plus en plus utilisé
- Langage implanté par de nombreux outils libres (Xalan, Saxon, ...)

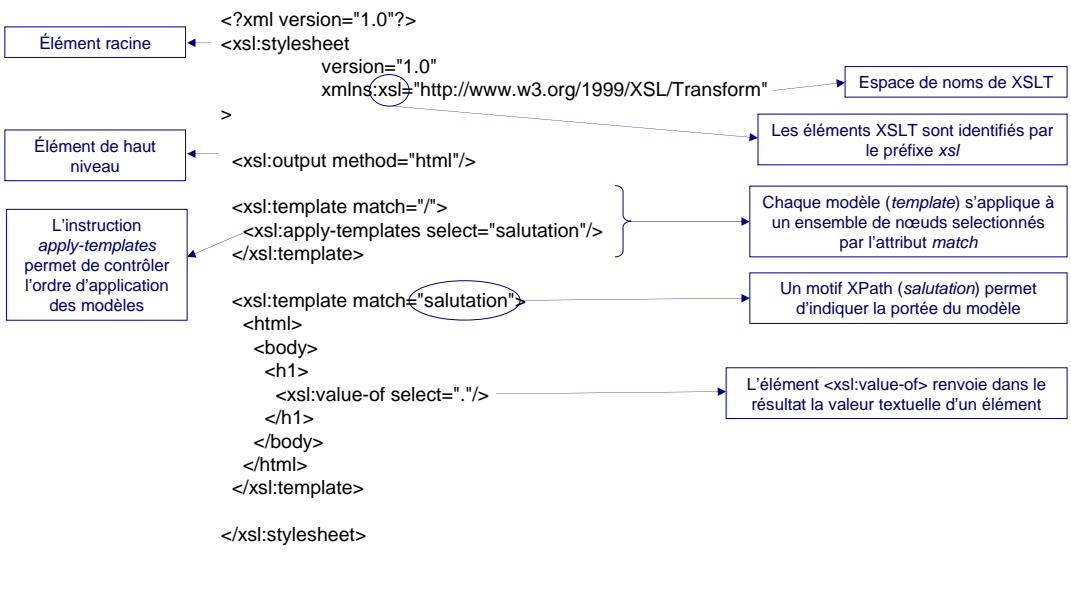
Inconvénients

- Pour la présentation Web, langage plus complexe que CSS
- La manière d'implanter des extensions au langage n'est pas encore normalisée

- Initiative exslt pour la normalisation des extensions (exslt.org).

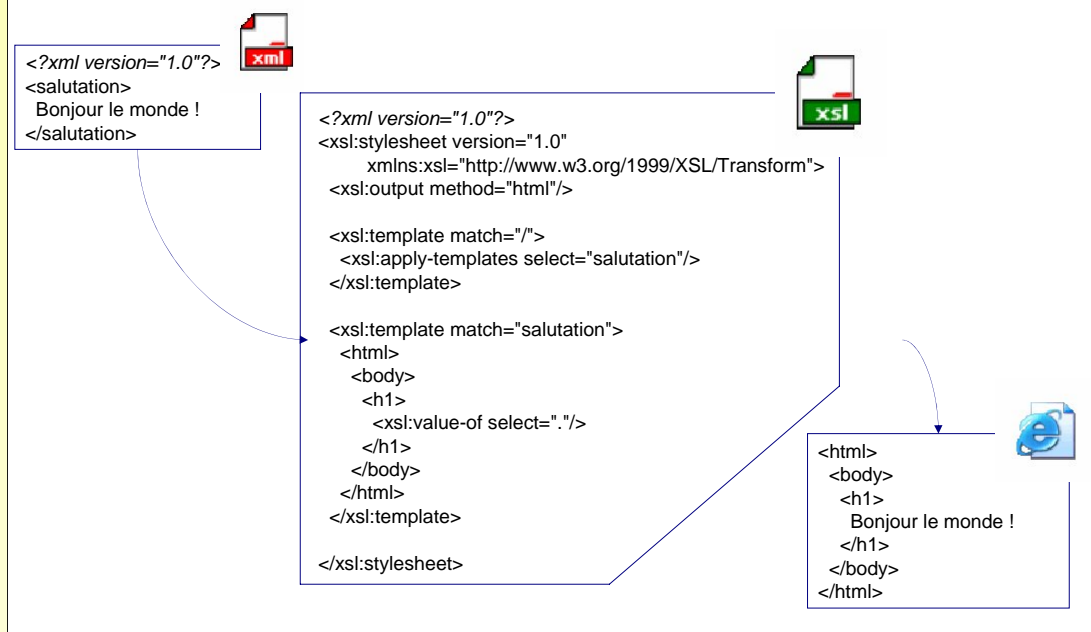
XSLT – Structure d'un document

□ Document XSLT



- Exemple : 1 – NS, 2 – préfixe, 3 – élément racine, 4 – élément de haut niveau, 5 – template, 6 – motif, 7 – apply-templates, 8 – xsl:value-of
- On peut remarquer que le langage est déclaratif, on écrit un modèle lié à un motif
- On aurait pu utiliser l'instruction transform au lieu de stylesheet
- Différence entre apply-template(s) et template : le apply-template(s) permet de constituer un ensemble de nœuds et pour chaque nœud, le processeur XSLT va chercher le template ayant un motif (match=« .. ») qui correspond et l'applique si tel est le cas
- Principe de parcours :
 - on prend en compte le nœud courant
 - on indique le chemin à suivre par rapport au nœud courant pour atteindre le (les) éléments à traiter par un template
- On peut remarquer une 1^o instruction XSLT utile, `<xsl:value-of>`, qui insère la valeur textuelle de l'élément dans l'arbre de sortie
- Pour les éléments de haut niveau, on a les éléments intéressants suivants :
 - `<xsl:output method="text" omit-xml-declaration="yes"/>`
 - `<xsl:strip-space elements="*" />`
 - `<xsl:include>` qui permet d'inclure un bout de document XSLT au document courant
 - `<xsl:import>` qui permet de redéfinir des déclarations XSLT contenus dans un autre document XSLT

XSLT – Exemple (1/4)

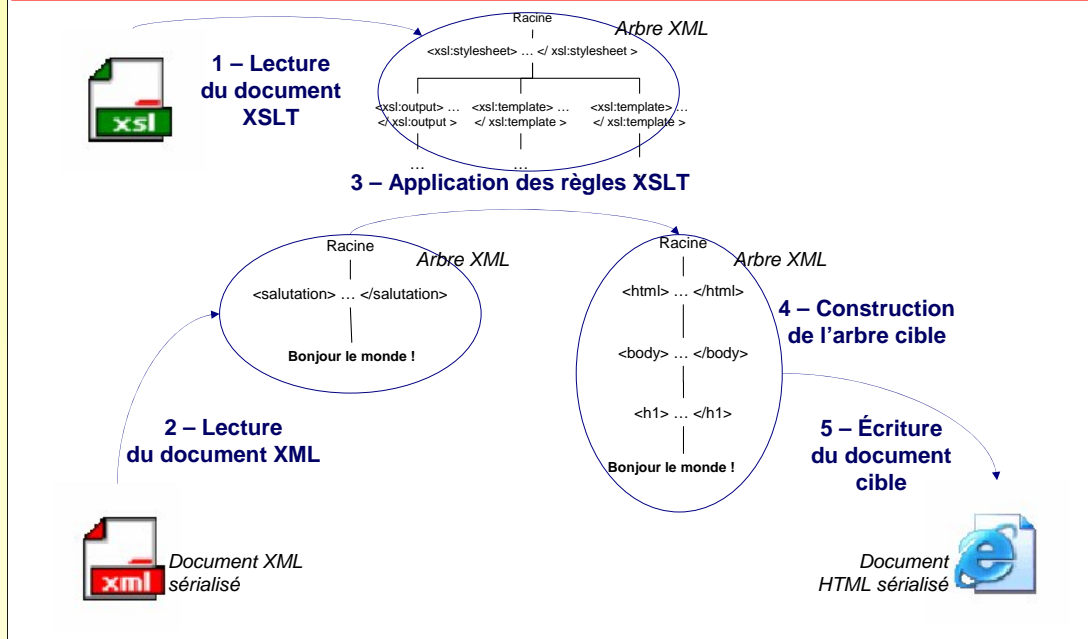


- 1° : on a un document XML.
- 2° : on écrit une feuille XSLT pour transformer le document XML.
- 3° : on obtient le document transformé en sortie (quand on génère du HTML avec XSLT, on obtient forcément du XHTML c'est-à-dire du HTML bien formé)
- **Remarque : pour les espaces de noms, on déclare le NS dans le prologue et on qualifie les éléments tels qu'ils sont dans le document source.**

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

<xsl:template match="dsi:personnes">
  <html>
  <head>
    <title>Ecrivains célèbres</title>
  </head>
  <body>
    <xsl:apply-templates />
  </body>
  </html>
</xsl:template>
```

XSLT – Exemple (2/4)



- Le processeur XSLT commence son travail sur un arbre créé après parsing du document source donc les éventuelles sections DOCTYPE, ENTITY ou CDATA (CDATA=section non-analysée lors du parsing par rapport à la syntaxe XML) ne sont pas considérées comme des nœuds particuliers.
- Le processeur XSLT parcourt l'arbre de gauche à droite et en profondeur d'abord.
- Ligne de commande utile pour réaliser cette transformation avec Xalan :
`java org.apache.xalan.xslt.Process -IN source.xml -XSL feuille.xsl -OUT cible.htm`

□ **Détail du processus de transformation réalisé par le processeur XSLT :**

- **Lecture (« parsing ») du document XSLT pour en construire une représentation arborescente en mémoire**
- **Parsing du document XML source pour en construire une représentation arborescente en mémoire**
- **Positionnement sur le nœud racine de l'arbre correspondant au document XML source et recherche d'un modèle dédié à cet élément (*match=/'*)**

```
<xsl:template match="/">  
  <xsl:apply-templates select="salutation"/>  
</xsl:template>
```

- **Exécution des instructions contenues dans le modèle courant**

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

□ **Détail du processus de transformation réalisé par le processeur XSLT (suite) :**

- Pour tous les éléments *salutation* contenus dans le nœud racine (élément courant), recherche et application du modèle dédié (ici *salutation* est forcément unique car c'est l'élément racine)

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

- A l'intérieur du modèle `<xsl:template match="salutation">`, les éléments non prefixés par *xsl* sont inclus dans l'arbre résultat tels quels (*html*, *body*, *h1*) et l'instruction `<xsl:value-of select="."/>` demande d'insérer dans l'arbre résultat la valeur du nœud de texte de l'élément courant (*salutation*)
- Une fois tous les éléments *salutation* traités (un seul dans l'exemple), retour dans le modèle dédié à l'élément racine
- Tous les éléments ont été traités donc le processeur XSLT écrit ("séréalise") l'arbre résultat dans le fichier de sortie

- Il existe des options d'exécution de Xalan permettant de voir comment les modèles sont appliqués :

-TT trace d'appel des modèles

-TTC trace d'appel des modèles enfant

- En résumé, un programme XSLT est une application successive d'un ensemble de règles, chacune concourant à la création du résultat final par assemblage de fragments qui viennent s'associer pour former une hiérarchie.

Document XML source

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="application/xml" href="personnes.xsl"?>
<personnes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="personnes.xsd">

  <personne>Alexandre Dumas</personne>
  <personne>Emile Zola</personne>

</personnes>
```

- Pour information, le schéma XML associé au document XML présenté :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
Description : schema XML
Auteur : David ROUSSE, d.rousse@wanadoo.fr
Date derniere modification : 08/05/2003
-->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<!-- Definitions types de donnees -->
<xsd:simpleType name="personneType">
  <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<xsd:complexType name="personnesType">
  <xsd:sequence>
    <xsd:element name="personne" type="personneType" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<!-- Definitions elements -->
<xsd:element name="personnes" type="personnesType" />

</xsd:schema>
```


Document XSLT

```
<?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="/">
    <xsl:apply-templates />
  </xsl:template>

  <xsl:template match="personnes">
    <html>
      <head><title>Ecrivains célèbres</title></head>
      <body>
        <xsl:apply-templates />
      </body>
    </html>
  </xsl:template>

  <xsl:template match="personne">
    <p><xsl:value-of select="."/;></p>
  </xsl:template>

</xsl:stylesheet>
```

Document HTML résultat de la transformation

```
<html>
  <head>
    <title>Ecrivains c&eacute;l&egrave;bres</title>
  </head>
  <body>
    <p>Alexandre Dumas</p>
    <p>Emile Zola</p>
  </body>
</html>
```

Le é et le è sont remplacés de manière automatique (par é et è) par le processeur XSLT Apache Xalan lors de la génération de documents HTML. Pour demander de manière explicite à ce que ce remplacement soit réalisé (par exemple pour un processeur XSLT dont le format de sortie par défaut n'est pas HTML), il faudra écrire dans le feuille XSLT :

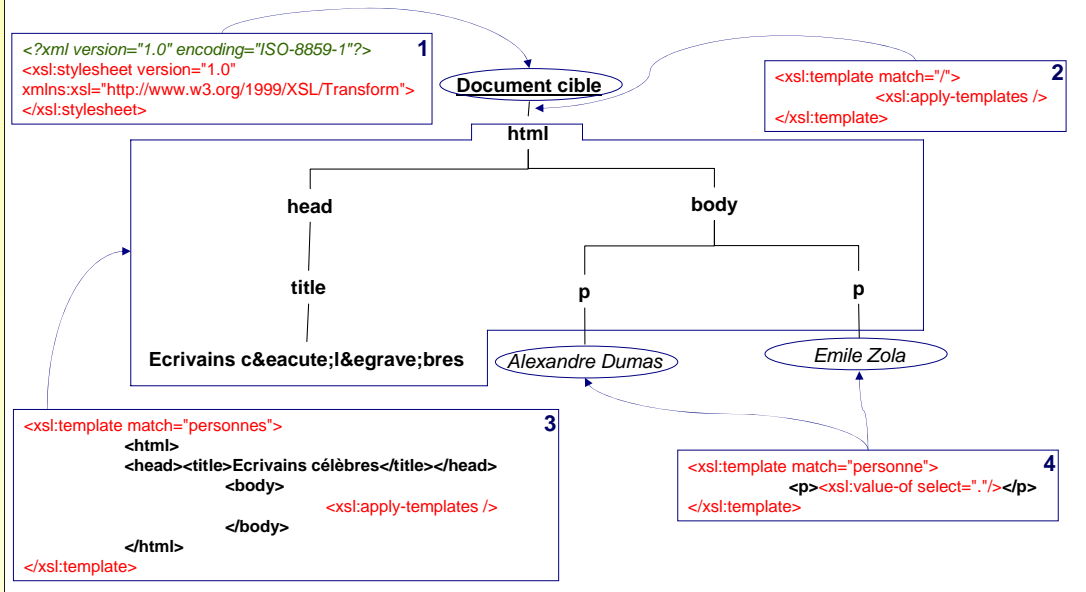
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="html"/>

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

...
```

□ Arbre cible que la processeur XSLT va construire



- 1 – Initialisation de l'environnement
- 2 – apply-templates du /
- 3 - apply-templates du personnes
- 4 - apply-templates du personne (en vrai 2 fois)
- 5.1 - apply-templates du personne (Dumas)
- 5.2 - apply-templates du personne (Zola)

Modèles par défaut pour l'élément racine et les éléments : permet de traiter tous les éléments du document XML

```
<xsl:template match="*/">  
  <xsl:apply-templates />  
</xsl:template>
```

```
<xsl:template match="*/" mode="x">  
  <xsl:apply-templates mode="x" />  
</xsl:template>
```

Modèle par défaut pour les éléments texte et les attributs :

```
<xsl:template match="text()|@">  
  <xsl:value-of select="." />  
</xsl:template>
```

Modèle par défaut pour les commentaires et les instructions de traitement :

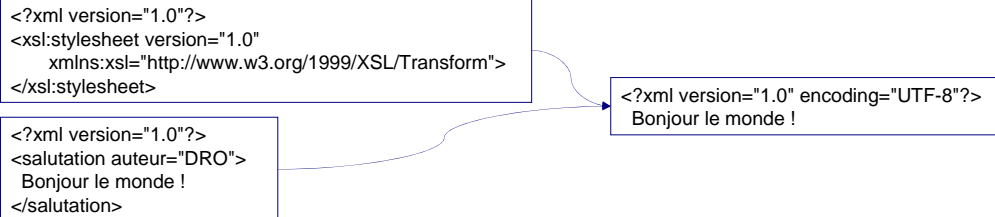
```
<xsl:template match="processing-instruction()|comment()" />
```

- Il existe 7 types de nœuds dans un document XML : texte, attributs, élément racine, élément, commentaire, instruction de traitement, espace de noms.
- Pour chaque type de nœud, XSLT fournit un modèle par défaut qui sera appliqué dans le cas où le concepteur de la feuille XSLT n'a pas fourni d'instruction(s) de traitement spécifique(s).
- Certains modèles assurent un traitement pour tout les éléments du document, d'autres permettent d'afficher le « contenu » (le nœud Text ou la valeur d'un attribut) et d'autres ne font « rien ».

Modèle par défaut pour les espaces de noms :

```
<xsl:template match="namespace()" />
```

-
- Attention**
- : le processeur XSLT considère qu'un élément est le père de ses attributs
- mais les attributs d'un élément ne sont pas ses fils. Donc, les attributs d'un document XML ne sont pas atteints via les règles par défaut**



- **XSLT a été pensé pour faciliter le traitement des nœuds d'éléments, les attributs servant à stocker des méta-informations.**
- **Cette feuille permettra d'obtenir l'affichage des nœuds texte ET attributs :**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>
<!-- traitement de tous les elements (racine et ses descendants) -->
<xsl:template match="*" />
  <xsl:apply-templates />
</xsl:template>
<!-- traitement de tous les elements -->
<xsl:template match="*">
  <!-- traitement de tous les noeuds fils de l'element -->
  <xsl:apply-templates />
  <!-- comme les attributs ne sont pas les fils de l'element qui les porte -->
  <!-- (par contre l'element qui a des attributs en est le pere) -->
  <!-- il faut ajouter cette ligne pour afficher la valeur des attributs -->
  <xsl:apply-templates select="@*" />
</xsl:template>
<xsl:template match="text()|@*">
  <xsl:value-of select="." />
</xsl:template>
</xsl:stylesheet>
```

□ Le document XML suivant servira de base pour la présentation des instructions XSLT

Instruction de traitement permettant d'associer une feuille XSLT au document XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="application/xml" href="personnes.xml"?>
<personnes>
  <personne naissance="1802" mort="1870">
    <nom>
      <prenom>Alexandre</prenom>
      <nom_famille>Dumas</nom_famille>
    </nom>
    <profession>Ecrivain</profession>
  </personne>
  <personne naissance="1840" mort="1902">
    <nom>
      <prenom>Emile</prenom>
      <nom_famille>Zola</nom_famille>
    </nom>
    <profession>Ecrivain</profession>
    <profession>Chroniqueur</profession>
  </personne>
</personnes>
```

- **Syntaxe de l'instruction de traitement permettant d'associer un document XML à une feuille XSLT :**

```
<?xml-stylesheet type="application/xml" href="personnes.xml"?>
```

Avec cible de l'instruction: `xml-stylesheet`

Et parametres de l'instruction: `type="application/xml" href="personnes.xml"`

- **Avec Microsoft Internet Explorer, il faut utiliser `type="text/xml"` pour associer le document XML à sa feuille de style. Mais cela est une interprétation de la norme car l'IANA ne référence que les types MIME `text/xml` et `application/xml` (voir RFC 3023).**

□ L'élément `<xsl:if>`

```
<!-- on sélectionne que les personnes nées avant 1840 -->
<!-- c'est à dire avec l'attribut naissance < 1840 -->
<xsl:if test="@naissance &lt; 1840">
  <xsl:apply-templates select="nom"/>
</xsl:if>

<!-- on sélectionne que les personnes dont le nom commence par Z -->
<xsl:if test="starts-with(nom_famille,'Z')">
  <p>
    <xsl:value-of select="nom_famille"/>,
    <xsl:value-of select="prenom"/>
  </p>
</xsl:if>
```

- Un document XSLT doit être bien formé donc il faut écrire par exemple `<` au lieu de `<`
- Il n'existe pas de `<xsl:else>`

□ L'élément `<xsl:choose>`

```
<!-- en fonction du nombres de profession(s) de la personne, l'affichage diffère -->
<xsl:choose>
  <xsl:when test="count(profession)=0">
    <xsl:apply-templates select="nom" />
    <br></br>
    <xsl:text>Sans profession</xsl:text>
  </xsl:when>
  <xsl:when test="count(profession)=1">
    <xsl:apply-templates select="nom" />
    <br></br>
    <xsl:text>Mono-activité</xsl:text>
  </xsl:when>
  <xsl:otherwise>
    <xsl:apply-templates select="nom" />
    <br></br>
    <xsl:text>Multi-compétences</xsl:text>
  </xsl:otherwise>
</xsl:choose>
```

- On peut remarque la branche `<xsl:otherwise>` dédié au traitement « par défaut ».

□ L'élément `<xsl:for-each>`

```
<xsl:template match="personnes">
<html>
  <head>
    <title>Ecrivains célèbres</title>
  </head>
  <body>
    <!-- traitement dans un boucle for-each des toutes les personnes -->
    <xsl:for-each select="personne">
      <xsl:apply-templates select="nom"/>
    </xsl:for-each>
  </body>
</html>
</xsl:template>
```

• Remarque :

- on aurait pu remplacer la boucle for-each par

...

```
<apply-templates select="personne" />
```

...

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

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

```
</xsl:template>
```

- on aurait aussi pu remplacer la boucle for-each par

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

□ L'élément `<xsl:call-template>`

```
<!-- définition d'un template dédié aux mentions légales -->
<!-- Noter que l'on utilise l'attribut name et non match -->
<!-- lors de l'écriture du template -->
<xsl:template name="copyright">
  <hr></hr>
  <font face="Arial" size="1"><i>
    &#xA9; Réalisé par David ROUSSE, rousse@dsi.cnrs.fr
  </i></font>
</xsl:template>

<xsl:template match="/">
  <xsl:apply-templates />
  <xsl:call-template name="copyright" />
</xsl:template>
```

- Le `<xsl:call-template>` appelle un modèle nommé `copyright`, défini précédemment.
- L'ordre de déclaration n'importe pas mais il est vivement conseillé de déclarer les modèles appelés avant les modèles appelants (même principe que l'`#include` entete.h dans un programme C par exemple).
- **ATTENTION** : à la différence de `apply-templates` ou d'un `for-each`, `call-template` ne change pas le nœud courant.

□ Les éléments <xsl:param> et <xsl:with-param>

```
<!-- definition d'un template avec un parametre -->
<xsl:template name="copyright">
  <xsl:param name="auteur" />
  <hr></hr>
  <font face="Arial" size="1"><i>
    <xsl:text>&#xA9; Réalisé par </xsl:text>
    <xsl:value-of select="$auteur" />
  </i></font>
</xsl:template>

<xsl:template match="/">
  <xsl:apply-templates />
  <xsl:call-template name="copyright">
    <xsl:with-param name="auteur" select="David ROUSSE" />
  </xsl:call-template>
</xsl:template>
```

□ L'élément <xsl:param> en tant que paramètre global

```
<!-- parametre global -->
<xsl:param name="OutputMode" />

<xsl:template match="/">
  <xsl:text>Valeur du parametre : </xsl:text>
  <!-- utilisation de la valeur du parametre OutputMode -->
  <xsl:value-of select="$OutputMode" />
</xsl:template>
```

• Passage de paramètre global à l'invite avec Apache Xalan :

```
java org.apache.xalan.xslt.Process -IN %1.xml -XSL %2.xsl -OUT %2.htm -param
nomParam valeurParam
```

□ L'élément `<xsl:variable>`

```
<xsl:template match="/">
  <!-- variable declaree contient "" par default -->
  <xsl:variable name="Candide" />
  <xsl:value-of select="$Candide" />

  <xsl:text>;</xsl:text>

  <!-- variable declaree et initialisee -->
  <xsl:variable name="Germinal" select="/personnes/personne[last()]/nom/nom_famille" />
  <xsl:value-of select="$Germinal" />

  <xsl:text>;</xsl:text>

  <!-- variable declaree avec valeur par default -->
  <xsl:variable name="MonteCristo">
    <xsl:text>Dumas</xsl:text>
  </xsl:variable>
  <xsl:value-of select="$MonteCristo" />
</xsl:template>
```

- Les variables en XSLT doivent être considérées comme l'équivalent de constantes (ou de variables statiques dans les langages de programmation classiques).

□ L'attribut mode de l'élément <xsl:template>

```
<xsl:template match="personnes" mode="html">
  <html>
    <head>
      <title>Ecrivains célèbres</title>
    </head>
    <body>
      <xsl:apply-templates select="personne" mode="html" />
    </body>
  </html>
</xsl:template>

<xsl:template match="personnes" mode="csv">
  <xsl:apply-templates select="personne" mode="csv" />
</xsl:template>

<xsl:template match="personne" mode="html">
  <xsl:value-of select="nom/nom_famille"/>
  <br></br>
</xsl:template>

<xsl:template match="personne" mode="csv">
  <xsl:value-of select="nom/nom_famille"/>
  <xsl:text>;</xsl:text>
</xsl:template>
```

- Un mode peut être vu comme une variante d'un template.
- Avec Apache Xalan, on peut appeler cette feuille XSLT en donnant à un paramètre global déclaré en tête de la feuille XSLT la valeur html ou csv selon le format de sortie désiré (ci-dessous, le nom du paramètre est OutputMode et sa valeur est html) :

```
java org.apache.xalan.xslt.Process -IN %1.xml -XSL %2.xsl -OUT %2.htm -param
OutputMode html
```

□ L'élément `<xsl:sort>`

```
<!-- traitement de toutes les personnes -->
<!-- par ordre decroissant de leurs dates de naissance -->
<xsl:apply-templates select="personne">
  <xsl:sort select="@naissance" order="descending" />
</xsl:apply-templates>

<!-- traitement de toutes les personnes -->
<!-- par ordre croissant de leurs dates de naissance -->
<xsl:for-each select="personne">
  <xsl:sort select="@naissance" />
  <xsl:apply-templates select="." />
</xsl:for-each>
```

- `<xsl:sort>` est toujours le fils d'un `<xsl:for-each>` ou d'un `<xsl:apply-template>`
- Il existe aussi 2 instructions pour copier tel quel un ou des nœuds source dans l'arbre cible :

`<xsl:copy>` copie le nœud dans l'arbre de sortie

`<xsl:copy-of>` copie un nœud et ses descendants dans l'arbre de sortie

- L'exemple de document XML suivant servira de base pour la présentation des fonctions XSLT id() et key()

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE personnes [
  <!ELEMENT personnes (personne*)>
  <!ELEMENT personne (nom)>
  <!ELEMENT nom (#PCDATA)>
  <!ATTLIST personne id ID #REQUIRED
    age CDATA #REQUIRED
    idlien IDREF #IMPLIED>
]>
<personnes>
  <personne id="n001" age="25" idlien="n003">
    <nom>Alexandre</nom>
  </personne>
  <personne id="n002" age="35" idlien="n003">
    <nom>Emile</nom>
  </personne>
  <personne id="n003" age="45">
    <nom>Honoré</nom>
  </personne>
</personnes>
```


□ La fonction id()

```
<xsl:template match="personne">
  <a name="{@id}"/>
  <xsl:apply-templates select="nom"/>
  <xsl:text> (</xsl:text><xsl:value-of select="@age"/><xsl:text> ans) </xsl:text>
  <!-- id() renvoie le noeud ayant l'ID passé en parametre -->
  <xsl:if test="{id(@idlien)}" >
    <a href="{id(@idlien)}/@id">
      <xsl:text>Voir aussi</xsl:text>
    </a>
  </xsl:if>
  <hr/></hr>
</xsl:template>

<xsl:template match="nom">
  <p>
    <xsl:text>Nom : </xsl:text><xsl:value-of select="."/>
  </p>
</xsl:template>
```

- **Il existe d'autres fonctions XSLT que id(), key() et document(). Reportez-vous à la documentation XSLT pour la liste exhaustive.**
- **Inconvénients de la fonction id() :**
 - Lecture de la DTD obligatoire pour que les éléments ID soient considérés comme des ID
 - Les liens d'intégrité sont stockés à l'intérieur du document (et non à l'extérieur comme en relationnel par ex.)
 - 1 attribut de type ID maximum par élément
 - id() ne retourne qu'un élément
 - 1 seul ensemble d'IDs est construit pour tout le document XML (les ID de personnes sont mélangés avec ceux de adresses par exemple)
 - la valeur des ID/IDREF doit être un nom XML valide (donc commencer par une lettre !)
- **Explication du template personne :**
 - Création d'un ancre
 - Application du modèle nom
 - Écriture de l'age entre parenthèses
 - S'il existe dans le document un nœud avec un attribut id dont la valeur est égale à l'attribut idlien du nœud courant, on entre dans le xsl:if et on crée un lien vers l'élément
- **La fonction generate-id() est disponible pour créer de manière automatique des identifiants en XSLT.**

□ La fonction key()

```
<!-- creation d'un index sur l'attribut age de l'element personne -->
<xsl:key name="age-index" match="/personnes/personne" use="@age" />
<!-- parametre global -->
<xsl:param name="age" />

<xsl:template match="personnes">
<html>
  <head>
    <title>Ecrivains célèbres</title>
  </head>
  <body>
    <!-- selectionne des noms correspondant au critere passe en parametre -->
    <!-- en utilisant l'index cree -->
    <xsl:apply-templates select="key('age-index', $age)" />
  </body>
</html>
</xsl:template>
```

• Passage du paramètre age avec Xalan :

```
java org.apache.xalan.xslt.Process -IN personnes.xml -XSL key.xsl -OUT key.htm -param
age 35
```

- Intérêt de key() : créer des index comme dans un SGBDR pour pouvoir ensuite faire des recherches du type `select * from personnes where @age=« valeur »` avec un index sur age.

□ La fonction document()

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet
  type="application/xml" href="document.xsl"?>
<personnes>
  <personne fichier="n001.xml" />
  <personne fichier="n002.xml" />
  <personne fichier="n003.xml" />
</personnes>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<personne id="n001" age="25" idlien="n003"><nom>Alexandre</nom></personne>
```

n001.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<personne id="n001" age="25" idlien="n003"><nom>Emile</nom></personne>
```

n002.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<personne id="n001" age="25" idlien="n003"><nom>Honoré</nom></personne>
```

n003.xml

document.xsl

```
...
<xsl:template match="personne">
  <personne>
    <nom><xsl:value-of select="document(@fichier)/personne/nom"/></nom>
    <age><xsl:value-of select="document(@fichier)/personne/@age"/></age>
  </personne>
</xsl:template>
...
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<personnes>
  <personne><nom>Alexandre</nom><age>25</age></personne>
  <personne><nom>Emile</nom><age>35</age></personne>
  <personne><nom>Honoré</nom><age>45</age></personne>
</personnes>
```

- Jusqu'à présent, on a eu un seul document XML source. La fonction document() permet d'utiliser plusieurs source XML.
- Dans cet exemple, la fonction document permet, dans une feuille XLST, de lire des données d'un document XML depuis un autre document XML.

□ Il est possible d'écrire des programmes XSLT, en utilisant une approche récursive

```
<xsl:template name="factorielle">
  <xsl:param name="parametre" />
  <xsl:choose>
    <xsl:when test="$parametre > 0">
      <xsl:variable name="n" select="$parametre" />
      <xsl:variable name="n_moins_un">
        <xsl:call-template name="factorielle">
          <xsl:with-param name="parametre" select="$parametre - 1" />
        </xsl:call-template>
      </xsl:variable>
      <xsl:value-of select="$n * $n_moins_un" />
    </xsl:when>
    <xsl:otherwise>1</xsl:otherwise>
  </xsl:choose>
</xsl:template>
<xsl:template match="/">
  <xsl:call-template name="factorielle">
    <xsl:with-param name="parametre" select="5" />
  </xsl:call-template>
</xsl:template>
```

Appel du modèle nommé

Appel récursif du modèle

- Rappel de l'algorithme récursif de calcul de factorielle :

```
factorielle(n) {
  Si(n>0)
    retourner n * factorielle(n-1)
  Sinon
    retourner 1
}
```

- Par opposition, l'écriture itérative de programmes est difficile en XSLT.

- Il est par ailleurs « facile » de se retrouver dans une boucle infinie en écrivant du XSLT (certains processeurs XSLT sont capables de les détecter).

□ L'extension d'élément

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:redirect="org.apache.xalan.xslt.extensions.Redirect"
  extension-element-prefixes="redirect">
  ...

  <xsl:template match="personnes">
    <xsl:if test="element-available('redirect:write')">
      <!-- autre test if test="contains(system-property('xsl:vendor'),'Apache') -->
      <xsl:for-each select="personne">
        <!-- ouverture fichier -->
        <redirect:write select="concat(@id, '.xml')">
          <nom>
            <xsl:value-of select="nom" />
          </nom>
        <!-- gestion des erreurs de traitement -->
        <xsl:fallback>
          <xsl:value-of select="nom" /><xsl:text> : erreur de traitement ! </xsl:text>
        </xsl:fallback>
        <!-- fermeture fichier -->
        </redirect:write>
      </xsl:for-each>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<personnes>
  <personne id="n001" age="25" idlien="n003">
    <nom>Alexandre</nom>
  </personne>
  <personne id="n002" age="35" idlien="n003">
    <nom>Emile</nom>
  </personne>
  <personne id="n003" age="45">
    <nom>Honoré</nom>
  </personne>
</personnes>
```

n001.xml

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

n002.xml

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

n003.xml

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

- Cet exemple permet de voir comment générer plusieurs fichiers XML en sortie d'une transformation.
- On déclare l'espace de noms particulier redirect, particulier car Xalan essaie de charger la classe Redirect (dans notre cas).
- L'instruction suivante extension-element-prefixes="redirect" indique au processeur de considérer les éléments préfixés par redirect comme des éléments d'extension.
- On teste si la fonction write est disponible dans la classe org.apache.xalan.xslt.extensions.Redirect.
- On utilise ensuite l'extension via redirect:write pour écrire dans le fichier de sortie.
- L'élément xsl:fallback permet de traiter les éventuels problèmes d'appel de redirect:write.

❑ L'extension d'élément pour l'appel de code personnalisé

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:cnrs="xalan://helloworld.HelloWorld"
  extension-element-prefixes="cnrs">

  <xsl:output omit-xml-declaration="yes"/>

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

  <xsl:template match="personnes">
    <xsl:if test="function-available('cnrs:sayHello')">
      <xsl:text>Resultat : </xsl:text>
      <xsl:value-of select="cnrs:sayHello(string(personne[ @id='n001']/nom))" />
    </xsl:if>
  </xsl:template>

</xsl:stylesheet>
```

- Ce code appelle la fonction sayHello(String nom) de la classe helloworld.HelloWorld.
- **ATTENTION** : pour faire fonctionner ce code avec Xalan, il faut mettre dans le CLASSPATH bsf.jar ainsi que HelloWorld.class
- Il est également possible d'ajouter ou d'appeler des fonctions, on parle alors d'extensions de fonctions.

- XSLT est puissant et extensible**
- L'écriture de feuilles XSLT doit suivre des règles strictes pour faciliter la maintenance**
- XSLT est la solution pour transformer du XML en un autre format, utilisable par une application ou lisible par l'utilisateur**
- XSLT peut être utilisé dans des domaines divers :**
 - Production de documents texte, hypertexte et XML
 - Échange et intégration de données
 - Production de documents destinés à l'impression
 - Publication de base de données

- **Durée prévue [Début – XSLT] : 1h10 minutes. Durée prévue [CSS – XSLT] : 1h.**
- **Exemples de règles d'écriture de feuille XSLT : pour les documents XML orienté données, écrire un template avec plusieurs for-each à l'intérieur, et écrire des templates pour chaque élément dans le cas de documents narratifs**
- **Production de documents texte, hypertexte et XML grâce à XSLT qui permet de convertir une document respectant un DTD/Schéma en un autre document lié à une autre DTD/Schéma.**
- **Échange et intégration de données dans la couche transformation dans les outils d' EAI par exemple.**
- **Production de documents destinés à l'impression via un contenu XML transformé en un format spécifique d'un application bureautique par exemple.**
- **Publication de base de données via l'extension Cocoon/XSP de Tomcat par exemple :**
 - Cocoon permet d'utiliser des pages XSD (eXtentible Server Pages) dont le principe est de dissocier logique de présentation et contenu à présenter dans des pages XML appelées pages XSD. Ces pages sont générées depuis le contenu XML par du XSLT et interprétées (à l'instar du ASP/JSP) par Cocoon pour générer du HTML.
 - Avec Cocoon et XSP, on aura les données en XML, la logique en XSP (il est recommandé d'utiliser des balises dynamiques pour externaliser le code du fichier XSP) et présentation via XSLT en HTML.

Introduction

Transformations de documents XML

Principes généraux

CSS

XSLT

XSL-FO

XPath

Liaisons de documents en XML

Principes généraux

XLink

XPointer

Manipulations de documents XML

Principes généraux

SAX

DOM

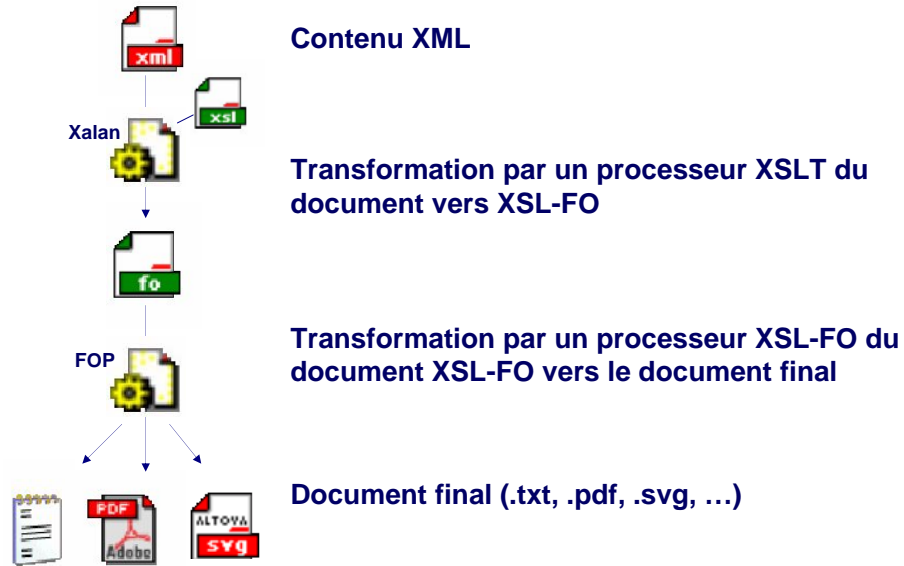
Conclusion

Références

- Langage XML spécifié par le W3C
- Fournit un ensemble d'instructions pour décrire la mise en forme de données stockées en XML
 - Introduit les notions de mise en page telles que les marges, l'enchaînement des pages, la notion de paragraphe, ...
- Conçu de manière généraliste pour prendre en compte
 - Des supports différents (papier, imprimante, ...)
 - Des conventions d'écritures différentes (gauche à droite, droite à gauche, ...)
 - Des formats différents (lettre, feuille, ...)

- Spécifications de la version 1.0 disponible à l'adresse <http://www.w3.org/TR/xsl/>
- XSL-FO permet d'exprimer des règles de mise en page.

- Dédié à la mise en forme du contenu de documents XML



- Outils implantant XSL-FO :
 - FOP d'Apache
 - XEP de la société RenderX
 - XSL Formatting Objects Composer d'IBM
 - XSL Formatter de Antenna House
 - Passive TEX de Sebastian Rahtz

XSL-FO – Exemple (1/3)

□ La feuille XSLT permettant de générer une feuille XSL-FO

```

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

<xsl:template match="/">
  <fo:root>
    <fo:layout-master-set>
      <fo:simple-page-master
        master-name="page_cv"
        page-height="29.7cm" page-width="21cm"
        margin-top="1cm" margin-bottom="1cm"
        margin-left="2.5cm" margin-right="2.5cm">
        <fo:region-body margin-top="1cm" margin-bottom="2cm" />
      </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-reference="page_cv">
      <fo:flow flow-name="xsl-region-body">
        <fo:block font-style="italic" font-family="serif" font-size="30pt" text-align="center" space-before="100pt">
          <fo:block><xsl:value-of select="salutation" /></fo:block>
        </fo:block>
      </fo:flow>
    </fo:page-sequence>
  </fo:root>
</xsl:template>
</xsl:stylesheet>

```

Espace de noms de XSL-FO

Élément racine du document XSL-FO à construire

Construction du document XSL-FO

Définition de la mise en page dans le(s) page-masters : on y décrit les différentes pagination de page dont a on besoin (introduction, corps, ...)

Éléments inclus dans <simple-page-master> servant à indiquer les différentes régions de la page (entête, corps, pied, ...)

Paragraphe

- **Exemple : 1 – NS, 2 – root, 3 – mise en page, 4 – body, 5 – page-sequence, 6 – block**
- **Principe : quand on écrit une feuille XSLT, il faut connaître la syntaxe du document résultat attendue (XSL-FO ici).**
- **Document XML source :**

```

<?xml version="1.0"?>
<salutation>
  Bonjour le monde !
</salutation>

```
- **Dans un <layout-master-set>, on peut aussi définir plusieurs simple-page-master et alors indiquer comme ces pages vont s'enchaîner, par exemple :**

```

<fo:page-sequence-master master-name="miseEnPage" >
  <fo:repeatable-page-master-alternatives>
    <fo:conditional-page-master-reference master-reference="page_introduction"
      page-position="first" />
    <fo:conditional-page-master-reference master-reference="page_corps"
      page-position="rest" />
  </fo:repeatable-page-master-alternatives>
</fo:page-sequence-master>

```

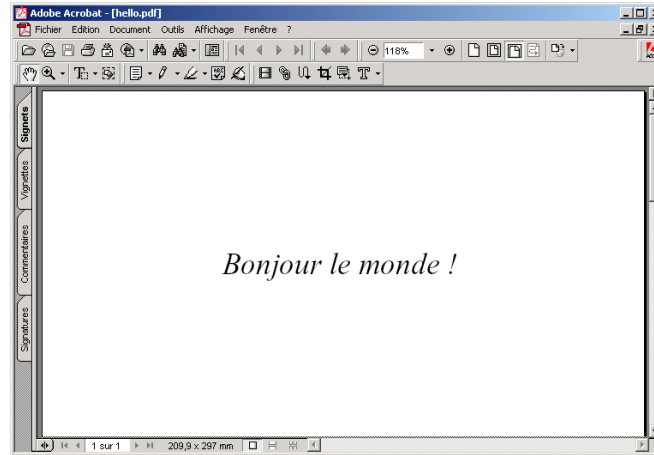
□ La feuille XSL-FO obtenue après l'exécution du processeur XSLT

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master margin-right="2.5cm" margin-left="2.5cm"
      margin-bottom="1cm" margin-top="1cm" page-width="21cm" page-
      height="29.7cm" master-name="page_cv">
      <fo:region-body margin-bottom="2cm" margin-top="1cm"/>
    </fo:simple-page-master>
  </fo:layout-master-set>
  <fo:page-sequence master-reference="page_cv">
    <fo:flow flow-name="xsl-region-body">
      <fo:block space-before="100pt" text-align="center" font-size="30pt" font-
      family="serif" font-style="italic">
        <fo:block>Bonjour le monde !</fo:block>
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

- **La feuille XSL-FO contient le résultat de l'interprétation des instructions XSLT de la feuille présentée dans le transparent précédent.**
- **Transformation via XSLT (Xalan) de documents XML vers XSL-FO :**
java org.apache.xalan.xslt.Process -IN source.xml -XSL feuille_xslt.xml -OUT source.fo
- **On remarque les principes généraux de XSL-FO ici :**
 - à chaque Formatting Object (chapitre, page, séquence) sont associées des propriétés de mise en page, de typographies, ...
 - les FO sont contenues dans les rectangles (les régions telles que region-body), eux même contenus dans d'autres rectangles que le processeur XSL-FO agence en fonction des règles de mise en page écrites par le programmeur ou implantées par défaut dans le processeur.

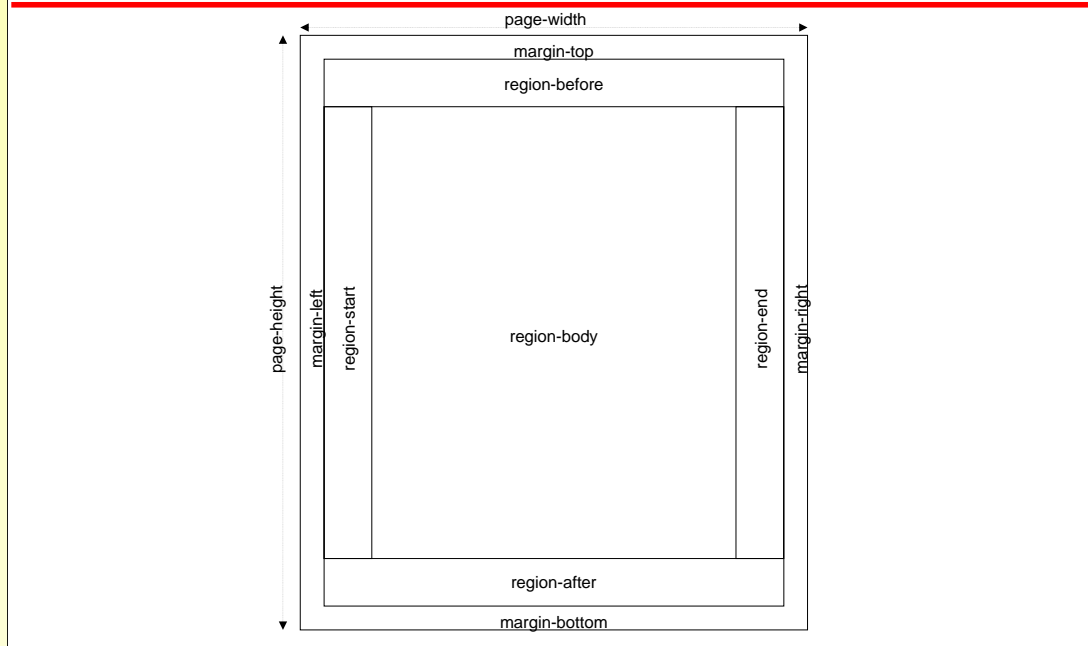
XSL-FO – Exemple (3/3)

- Le document PDF obtenu après l'exécution du processeur FOP



- On obtient ainsi un document PDF « gratuitement » (sans avoir à utiliser de produits commerciaux comme Adobe Distiller).
- Transformation via FOP de documents XSL-FO vers PDF :
`java org.apache.fop.apps.Fop -fo source.fo -pdf source.pdf`

XSL-FO – Modèle de mise en page



- Code XSLT permettant d'adresser les régions présentées ci-dessus :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:fo="http://www.w3.org/1999/XSL/Format">
<xsl:template match="/">
<fo:root>
  <fo:layout-master-set>
    <fo:simple-page-master master-name="page_cv"
      page-height="29.7cm"
      page-width="21cm"
      margin-top="1cm"
      margin-bottom="1cm"
      margin-left="2.5cm"
      margin-right="2.5cm">
      <fo:region-body margin-top="1cm" margin-bottom="2cm"/>
      <fo:region-before extent="1.5cm" />
      <fo:region-after extent="1.5cm" />
      <fo:region-start extent="1.5cm" />
      <fo:region-end extent="1.5cm" />
    </fo:simple-page-master>
  </fo:layout-master-set>
  <fo:page-sequence master-reference="page_cv">
    <fo:static-content flow-name="xsl-region-before">
      <fo:block text-align="center"
        font-size="9pt"
        font-family="serif"
        font-style="italic">
        DSI - CNRS - REGION BEFORE
      </fo:block>
    </fo:static-content>
    <fo:static-content flow-name="xsl-region-after">
```

- Langage permettant de décrire la mise en page de contenu XML tout en s'affranchissant d'une application particulière
- Langage encore jeune (version 1.0 éditée en octobre 2001) donc peu implanté dans les applications actuelles
- Finalement, la situation est aujourd'hui la suivante :
 - CSS est adapté à la présentation Web
 - XSL-FO est adapté à la mise en page de contenu complexe, par exemple pour l'impression
 - XSLT joue un rôle central dans la préparation de données XML pour d'éventuelles présentations avec CSS ou XSL-FO

- Apache FOP est aujourd'hui en version 0.20.5 donc est encore jeune ...
- Atelier de 10 minutes puis partie sur XPath.

Introduction

Transformations de documents XML

Principes généraux

CSS

XSLT

XSL-FO

XPath

Liaisons de documents en XML

Principes généraux

XLink

XPointer

Manipulations de documents XML

Principes généraux

SAX

DOM

Conclusion

Références

XPath – Définition

- ❑ Langage de requêtes non XML permettant d'identifier des nœuds contenus dans des documents XML
- ❑ Utilisé dans XSLT comme langage de référencement pour sélectionner des éléments du document à transformer
 - Exemple :
`<xsl:apply-templates select="personne" />`
- ❑ Utilisé dans les schémas XML pour définir des contraintes d'unicité et de référence
 - Exemple :
`<xsd:key name="pk_personne">
 <xsd:selector xpath="personne" />
 <xsd:field xpath="nom_famille" />
</xsd:key>`
- ❑ Utilisé dans XPointer pour identifier des parties contenues dans un document XML
 - Exemple :
`xpointer(/personnes/personne)`

- `<xsl:apply-templates select="personne" />` demande au processeur XSLT de déclencher le template relatif à tous les éléments `personne` fils du nœud courant.

- Le code suivant :

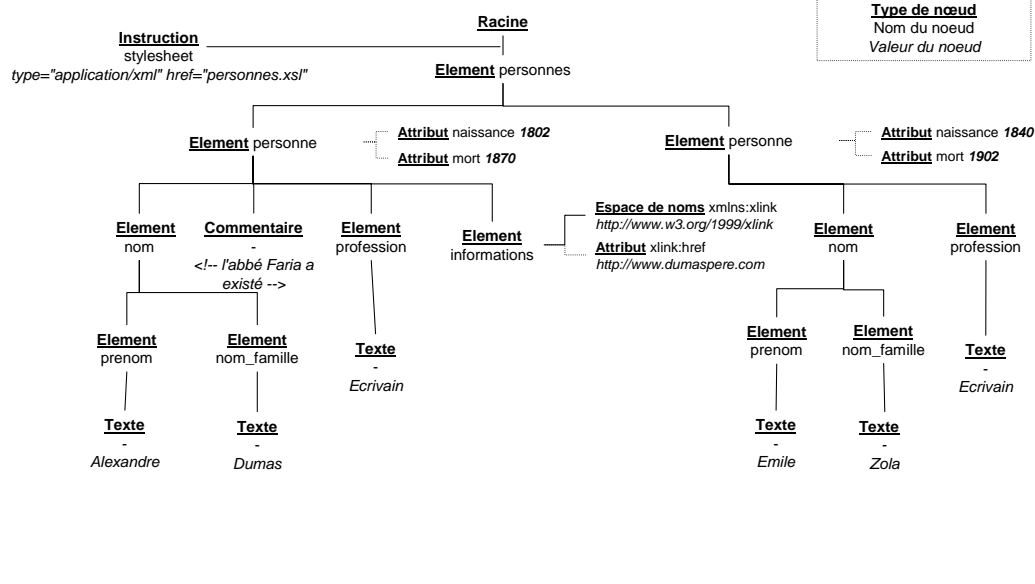
```
<xsd:key name="pk_personne">  
    <xsd:selector xpath="personne" />  
    <xsd:field xpath="nom_famille" />  
</xsd:key>
```

- crée une clé primaire appelée `pk_personne`
 - l'expression XPath écrite dans l'élément "selector" indique l'élément qui doit être clé (doit pointer un nœud de type élément).
 - le deuxième chemin XPath (évalué par rapport au nœud sélectionné) spécifie le nœud identifiant l'élément qui doit être unique, non nul et référençable (doit pointer un nœud de type élément ou attribut).
- L'expression XPointer `xpointer(/personnes/personne)` référence l'ensemble des éléments `personne` fils de `personnes`.

□ Document XML source

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="application/xml" href="personnes.xsl"?>
<personnes>
  <personne naissance="1802" mort="1870">
    <nom>
      <prenom>Alexandre</prenom>
      <nom_famille>Dumas</nom_famille>
    </nom>
    <!-- l'abbé Faria a existé -->
    <profession>Ecrivain</profession>
    <informations xmlns:xlink="http://www.w3.org/1999/xlink"
      xlink:href="http://www.dumaspere.com" />
  </personne>
  <personne naissance="1840" mort="1902">
    <nom>
      <prenom>Emile</prenom>
      <nom_famille>Zola</nom_famille>
    </nom>
    <profession>Ecrivain</profession>
  </personne>
</personnes>
```

□ Arbre correspondant au document XML précédent



- Exemple : 1 - /personnes/personne, 2 – nom/prenom/text()
- Le nœud Racine correspond à l'ensemble du document.
- Les attributs xmlns sont considérés comme des nœuds d'espace de noms (et non comme des attributs) et sont rattachés à tous les nœuds de l'élément ou d'attributs dans la portée de la déclaration.
- Un élément ne peut avoir qu'un seul élément Texte fils. Donc il existe une étape de normalisation qui fusionne les sections CDATA, les blancs et tout le texte en un seul nœud Texte fils d'un Element donné.
- Les éventuelles sections CDATA sont normalisées et n'apparaissent pas dans l'arbre.

☐ Le tableau suivant présente les 7 types de nœuds d'un arbre XML

Type de nœud	Nom du nœud	Valeur du nœud
Racine	-	-
Element	Nom de l'élément	-
Attribut	Nom de l'attribut	Valeur de l'attribut
Texte	-	Valeur du texte
Commentaire	-	Valeur du commentaire
Instruction de traitement	Nom de la cible	Paramètre(s) de la cible
Espace de noms	Préfixe de l'espace de noms	URI désignant l'espace de noms

- ❑ Une expression XPath est constituée d'une suite d'étapes, séparées par des /

[/étape₁/étape₂/.../étape_n

- ❑ Une expression XPath est évaluée à partir du nœud contexte, autrement dit à partir du nœud en cours de traitement dans l'arbre XML

- Lorsque l'expression XPath commence par / alors le nœud contexte est toujours la racine (on parle de chemin absolu).
- Lorsque l'expression XPath ne commence pas par / alors le nœud contexte change d'étape en étape (on parle de chemin relatif).

- ❑ Chaque étape peut se diviser en trois parties

L'axe définit le sens de parcours dans l'arbre à partir du nœud contexte

axe::filtre[prédicat₁] [prédicat₂]...

Le filtre indique le type de nœuds à retenir (élément, attribut, texte, ...)

Le(s) prédicat(s) exprime(nt) les conditions que doivent satisfaire les nœuds retenus dans le résultat de l'expression à l'issue du filtrage

- Par défaut, le nœud contexte est /
- Le nœud contexte est le nœud à partir duquel une expression XPath est évaluée. On peut le comparer au répertoire courant dans les systèmes de fichier.
- Plus généralement, le contexte est une structure qui contient l'ensemble des informations nécessaires à l'évaluation

d'une expression XPath : nœud contexte, 2 entiers indiquant la position du contexte et la taille du contexte (le numéro du nœud en train d'être traité et le nombre de nœuds à traiter), un ensemble de variables, un ensemble de fonctions XPath et XSLT utilisables dans la requête et un ensemble de déclarations d'espaces de noms.

Une syntaxe XPath **abrégée** existe (par opposition à la syntaxe de base appelée parfois syntaxe non abrégée), grâce à l'axe par défaut, child, et à la notation @ désignant l'axe attribute

En pratique, la syntaxe abrégée est la plus utilisée

Les expressions suivantes sont ainsi des expressions XPath

▪ Exemple 1 :

```
<!-- Syntaxe abrégée -->
personne/nom/nom_famille
<!-- Syntaxe non abrégée -->
child::personne/child::nom/child::nom_famille
```

▪ Exemple 2 :

```
<!-- Syntaxe abrégée -->
@naissance
<!-- Syntaxe non abrégée -->
attribute::naissance
```

- La syntaxe de base (appelée parfois syntaxe non abrégée) est à utiliser quand il est nécessaire de réaliser des parcours de l'arbre selon des axes précis (autre que child).
- La syntaxe non abrégée est puissante mais peu intuitive. A l'inverse, la syntaxe abrégée reprend les concepts de parcours de fichiers UNIX et est donc simple à comprendre.

Les filtres permettent d'éliminer des nœuds parmi ceux sélectionnés dans un axe

- Exemples :

```
<!-- Syntaxe abrégée -->  
personne/nom
```

```
<!-- Syntaxe non abrégée -->  
child::personne/child::nom
```

Les prédicats (expressions booléennes) sont constitués d'un ou plusieurs tests qui sont appliqués à chaque nœud dans la liste des nœuds résultante de l'application du filtre

- Exemples :

```
<!-- Syntaxe abrégée -->  
personne[@naissance='1802']
```

```
<!-- Syntaxe non abrégée -->  
child::personne[attribute::naissance='1802']
```

- L'expression XPath suivante sélectionne le nœud attribut dont la valeur est 1802 :
`child::personne/attribute::naissance[.='1802']`

- ❑ Le tableau suivant présente les 13 axes pour écrire une expression XPath

Axe	Définition	Axe	Définition
child	Fils du nœud contexte (axe par défaut)	following-sibling	Tous les frères droits du nœud contexte
attribute	Attributs du nœud contexte (abrégé en @)	preceding	Tous les nœuds précédents le nœud contexte dans l'ordre de parcours du document
parent	Père du nœud contexte	following	Tous les nœuds suivants le nœud contexte dans l'ordre de parcours du document
descendant	Tous les descendants du nœud contexte	descendant-or-self	Tous les descendants du nœud contexte et le nœud contexte
ancestor	Tous les ancêtres du nœud contexte	ancestor-or-self	Tous les ancêtres du nœud contexte et le nœud contexte
self	Le nœud contexte	namespace	Les nœuds d'espace de noms
preceding-sibling	Tous les frères gauches du nœud contexte		

Source : Comprendre XSLT, ISBN 2841771482

- ❑ L'axe attribute est particulier car il est le seul à s'appliquer à l'arbre des attributs, arbre qui a toujours pour racine un nœud de type Element, et un seul niveau constitué de feuilles non ordonnées identifiées par leur nom (nom de l'attribut).

- Les jokers facilitent l'écriture de filtres XPath (les jokers s'évaluent à partir du nœud courant)

Joker	Nœuds sélectionnés
*	Tous les nœuds d'éléments fils du nœud contexte
@*	Tous les nœuds d'attributs du nœud contexte
node()	Tous les types de nœud
text()	Tous les nœuds de type Texte
comment()	Tous les nœuds de type Commentaire
processing-instruction()	Tous les nœuds de type Instruction de traitement
.	Le nœud contexte
..	Le père du nœud contexte
//	Tous les nœuds descendants du nœud contexte

Source : Comprendre XSLT, ISBN 2841771482

- Il faut être conscient que l'utilisation de // implique une charge de travail importante pour le processeur XSLT (parcours de tous les descendants) et donc ce joker est à utiliser avec parcimonie.
- Il existe également un opérateur de sélection multiple (le OU booléen) noté |. Exemple :

```
<xsl:template match="personnes">
  <html>
    <head>
      <title>Ecrivains célèbres</title>
    </head>
    <body>
      <xsl:apply-templates />
    </body>
  </html>
</xsl:template>
<xsl:template match="nom_famille|profession">
  <p><i>
    <xsl:value-of select="." />
  </i></p>
</xsl:template>
```

- Pour sélectionner une instruction de traitement, on écrit :

```
<xsl:value-of select="/processing-instruction('xml-stylesheet')"/>
```

- Le tableau suivant compare la syntaxe non abrégée et son équivalente abrégée

Syntaxe non abrégée	Syntaxe abrégée
<code>self::node()</code>	<code>.</code>
<code>parent::node()</code>	<code>..</code>
<code>attribute::nom_attribut</code>	<code>@nom_attribut</code>
<code>attribute::*</code>	<code>@*</code>
<code>child::nom_element/attribute::nom_attribut</code>	<code>nom_element/nom_attribut</code>
<code>/child::nom_element</code>	<code>/nom_element</code>
<code>/descendant-or-self::node()</code>	<code>//</code>
<code>self::node()/descendant-or-self::node()</code>	<code>./</code>
<code>child::nom_elementA/ descendant-or-self::node()/ child::nom_elementB</code>	<code>nom_elementA//nom_elementB</code>

Source : Comprendre XSLT, ISBN 2841771482

- Un petit piège couramment rencontré : la confusion entre `.` et `current()`. Le `.` désigne est un raccourci XPath qui désigne le noeud contexte au sein même de l'expression XPath alors que `current()` est une fonction XSLT qui désigne le noeud courant au niveau de l'instruction XSLT. En général, `.` et `current()` désignent le même noeud :

`<xsl:value-of select="." />` et `<xsl:value-of select="current()" />` sont équivalentes la plupart du temps

- Par contre, dans une expression XPath « de parcours », `.` et `current()` diffèrent :

```
<xsl:variable name="lib">
```

```
<xsl:value-of select="./libelle"/>
```

```
</xsl:variable>
```

```
<xsl:for-each select="//idValeursCodes/code[@libelle=$lib]/choix">
```

peut s'écrire ainsi :

```
<xsl:for-each select="//idValeursCodes/code[@libelle=current()/libelle]/choix">
```

- Pour de plus amples informations sur les notions de noeud contexte et noeud courant, ne pas hésiter à contacter rousse@dsi.cnrs.fr

XPath – Exemples

□ Quelques exemples de requêtes XPath

- Exemple 1 :
<!-- Syntaxe abrégée -->
//@*
<!-- Syntaxe non abrégée -->
/descendant-or-self::node()/attribute::*

- Exemple 2 :
<!-- Syntaxe abrégée -->
personne/@naissance
<!-- Syntaxe non abrégée -->
child::personne/attribute::naissance

- Exemple 3 :
<!-- Syntaxe abrégée -->
personne[@naissance and position()=last()]
<!-- Syntaxe non abrégée -->
child::personne[attribute::naissance and position()=last()]

- **personne[@naissance and position()=last()]** peut s'écrire aussi **personne[@naissance][position()=last()]**
- **child::personne[attribute::naissance and position()=last()]** peut s'écrire aussi **child::personne[attribute::naissance][position()=last()]**

□ **Un modèle de valeur d'attribut indique au processeur XSLT d'évaluer l'expression XPath et d'insérer le résultat à la place du modèle**

- Un modèle de valeur d'attribut s'utilise ainsi en XSLT

```
<xsl:template match="informations">
  <a href="{@xlink:href}"><xsl:value-of select="@xlink:href"/></a>
</xsl:template>
```

- Le document résultat de la transformation XSLT est le bout de code suivant

```
<a href="http://www.dumaspere.com">
  http://www.dumaspere.com
</a>
```

XPath – Fonctions

 Fonctions sur les chaînes de caractères

```
substring(personne[position()=1]/nom/nom_famille,1,2)  
substring-before(personne[position()=1]/nom/nom_famille,'m')
```

 Fonctions sur les nombres

```
number(count(personne))  
number(personne[position()=last()]/@mort)-number(personne[position()=1]/@mort)
```

 Fonctions booléennes

```
boolean('hello')  
starts-with(personne[position()=1]/@naissance,'18')
```

- Pour information, il existe 4 types de données en XPath :
- Les **chaînes de caractères**, **string**, constituées d'une suite de caractères UNICODE, délimitées par des " " ou des ' '
- Les **nombres**, **number**, sont des nombres à virgule flottante codés sur 64 bits (type double de JAVA) que l'on peut manipuler avec les opérateurs +, -, *, div et mod
- Les **booléens** sont produits par les opérateurs de comparaison =, !=, <, >, <=, >= et les fonctions true() et false()
- Les **ensembles de nœuds** (0 à n éléments) sont produits par le résultat de requêtes XPath présentées précédemment

- **Remarque :**
 - l'erreur NaN avec les fonctions numériques signifie que le paramètre n'est pas un nombre valide (Not A Number)
 - Si on ne mentionne pas le prédicat [position()=1] dans `substring(personne[position()=1]/nom/nom_famille,1,2)`, le 1° élément personne est utilisé

- Pour une liste exhaustive des fonctions, voir <http://www.w3.org/>

- XSL (XSLT et XSL-FO) est par essence lié à XPath pour sélectionner des nœuds d'un document XML
- L'écriture de requêtes XPath se base sur une représentation arborescente d'un document XML
- La syntaxe XPath abrégée est intuitive et se rapproche de l'écriture d'instructions UNIX de parcours du système de fichier
- La syntaxe XPath non abrégée est puissante et permet d'adresser tous les nœuds d'un document XML
- Un groupe de travail du W3C est en train de spécifier un langage de requêtes pour des données stockées en XML, XML Query (l'équivalent du SQL pour les bases de données relationnelles)
- Une version 2.0 de XPath est en cours de spécification par le W3C

- **Durée prévue [Début – XPath] : 1h30 minutes. Durée prévue [XSLT – XPath] : 30 minutes.**
- **Pour s'entraîner à écrire des requêtes XPath, il existe l'utilitaire ApplyXPath fourni avec Apache Xalan.**
- **Certaines parties de XML Query 1.0 et XPath 2.0 sont traitées par le même « sous-groupe » de travail au sein du W3C**

Liaisons de documents en XML

Avril 2003

Direction des systèmes d'information

Introduction

Transformations de documents XML

Principes généraux

CSS

XSLT

XSL-FO

XPath

Liaisons de documents en XML

Principes généraux

XLink

XPointer

Manipulations de documents XML

Principes généraux

SAX

DOM

Conclusion

Références

-
- La problématique se situe autour de la manière d'exprimer des liens en XML
 - Écrire en XML l'équivalent d'une balise <a> en HTML par exemple
 - Étant donné la structure logique en arbre d'un document XML, la description de relations autres que parent-enfant(s) entre nœuds n'est pas naturelle
 - Seuls existent de manière native les références (ID/IDREF/IDREFS dans les DTD et <xsd:key>/<xsd:keyref> dans les schémas XML) pour lier des nœuds d'un même arbre XML
 - Seule existe de manière native la fonction document() en XSLT appliquée à un document XML contenant un index de noms de fichiers pour lier des nœuds d'arbres XML différents
 - Une problématique d'expression de la notion de liens en XML apparaît

Transformations de documents XML

Principes généraux

CSS

XSLT

XSL-FO

XPath

Liaisons de documents en XML

Principes généraux

[XLink](#)

XPointer

Manipulations de documents XML

Principes généraux

SAX

DOM

Références

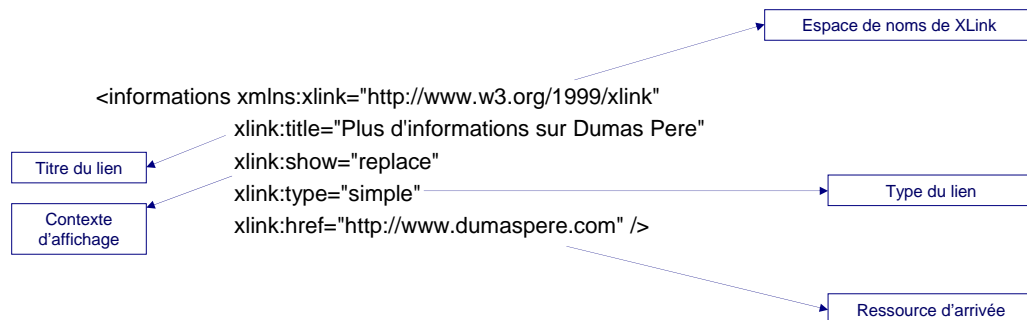
XLink – Définition

- Langage XML spécifié par le W3C
- Permet de décrire des graphes dans lesquels les sommets sont des documents (situés à des URI particuliers) et les arêtes les liens entre ces documents
- Fournit ainsi un vocabulaire puissant pour décrire
 - Des liens unidirectionnels pour réaliser l'équivalent de la balise <a> de HTML par exemple
 - Des liens multidirectionnels entre tout type de document (XML ou autres) pour réaliser des index et des tables des matières par exemple

- **Spécifications de la version 1.0 disponible à l'adresse <http://www.w3.org/XML/Linking>**
- **XLink signifie XML Linking Language**
- **URI = Uniform Resource Identifiers**

XLink – Exemple (1/2)

□ Lien unidirectionnel, entre une ressource de départ et une ressource d'arrivée



- Exemple : 1 – NS, 2 – title, 3 – show, 4 – type, 5 – href
- Pour information Mozilla 1.3 supporte les liens simples.
- L'attribut `xlink:show` peut prendre les valeurs suivants : `new`, `replace`, `embed`, `other` et `none`
- Il existe l'attribut `xlink:actuate` qui indique à l'application qui lit le document à quel moment charger le lien : `onLoad`, `onRequest`, `other`, `none`
- Pour exprimer la sémantique des liens, en plus de `xlink:title`, il existe `xlink:role` qui contient le texte décrivant la ressource distante
- Il existe l'attribut `xml:base` pour associer un espace de noms à un lien :

```
<xml:base="/www.picks.com/">
```

```
<item>
```

```
<link xlink:type="simple" xlink:href="pick.xml">Pick</link>
```

```
</item>
```

```
</list>
```

XLink – Exemple (2/2)

❑ Lien multidirectionnel de type arc

```
<series xlink:type="extended" xmlns:xlink="http://www.w3.org/1999/xlink">
  <auteur>E. Zola</auteur>
  <roman xlink:type="locator" xlink:label="ez1" xlink:href="ftp://archive.org/zola1.txt">
    <titre>La curée</titre>
  </roman>
  <roman xlink:type="locator" xlink:label="ez2" xlink:href="ftp://archive.org/zola2.txt">
    <titre>La fortune des Rougon</titre>
  </roman>
  <roman xlink:type="locator" xlink:label="ez3" xlink:href="ftp://archive.org/zola3.txt">
    <titre>L'assomoir</titre>
  </roman>
  <!-- arcs -->
  <suivant xlink:type="arc" xlink:from="ez1" xlink:to="ez2"/>
  <suivant xlink:type="arc" xlink:from="ez2" xlink:to="ez3"/>
  <precedent xlink:type="arc" xlink:from="ez2" xlink:to="ez1"/>
  <precedent xlink:type="arc" xlink:from="ez3" xlink:to="ez2"/>
</series>
```

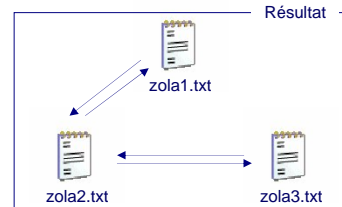
Espace de noms de XLink

Identifiant à utiliser lors de la
définition des liens de l'arc

Type du lien

Une flèche de l'arc (xlink:type="arc") avec pour
départ xlink:from="ez1" et arrivée xlink:to="ez2"

Arcs



- Exemple : 1 – NS, 2 – type, 3 – label, 4 – arc, 5 – flèche, 6 – resultat
- Ce qu'il faut voir c'est que l'on va utiliser un ensemble de balises prédéfinies (une application XML) pour répondre à un besoin : exprimer des liens de manière normalisé.

Introduction

Transformations de documents XML

Principes généraux

CSS

XSLT

XSL-FO

XPath

Liaisons de documents en XML

Principes généraux

XLink

XPointer

Manipulations de documents XML

Principes généraux

SAX

DOM

Conclusion

Références

XPointer – Définition

- Langage non XML spécifié par le W3C
- Dédié à l'identification de fragments de documents XML,
- Souvent utilisé en liaison avec XLink
- Utilise XPath pour localiser certains fragments de documents XML et implante de nouvelles fonctionnalités destinées à la localisation des points et des régions

▪ Exemples :

<!-- sélection de tous les éléments personne fils de l'élément personnes -->

<!-- du document www.dsi.cnrs.de/personnes.xml -->

[http://www.dsi.cnrs.fr/personnes.xml#xpointer\(/personnes/personne\)](http://www.dsi.cnrs.fr/personnes.xml#xpointer(/personnes/personne))

<!-- sélection du 2° fils de l'élément racine du document -->

[element\(/1/2\)](#)

- XPointer signifie XML Pointer Language. Il permet de localiser des fragments de documents (notamment de type text/xml ou application/xml). Plus précisément, il existe un framework XPointer autour duquel des schémas sont définis comme `element()`, `xmlns()` et `xpointer()`
- Le stade de Recommandation a été atteint le 25/03/2003 pour le framework Xpointer et les schémas `element()` et `xmlns()` (disponible à l'adresse <http://www.w3.org/TR/xptr/>). Le schéma `xpointer` est encore une Working Draft
- La notion d'espaces de noms est prise en compte par XPointer via le schéma `xmlns()` :

```
<customer xmlns="http://example.org/customer">
```

```
  <name xmlns="http://example.org/personal-info">John Doe</name>
```

```
</customer>
```

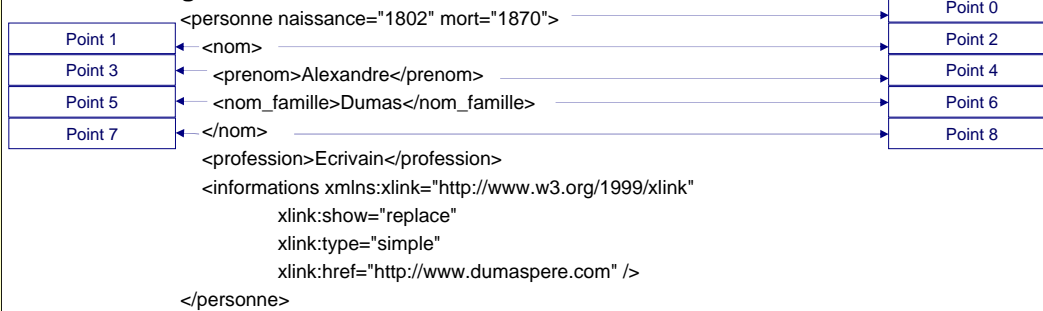
```
xmlns(c=http://example.org/customer)
```

```
xmlns(p=http://example.org/personal-info)
```

```
xpointer(/c:customer/p:name)
```

XPointer – Exemple

□ Points et régions



- Sélection du point situé après </prenom> (le point 4) :

```
xpointer(end-point(//prenom))
```

- Sélection de la région couvrant l'élément <prenom> (entre les points 3 et 4) :

```
xpointer(range(//prenom))
```

- L'intérêt est ainsi de pouvoir sélectionner tout ce que l'on veut dans un document XML sans avoir à modifier ce dernier (sans avoir à y ajouter des ancres ou autres par exemple)
- **ATTENTION** : pour les besoins de la présentation, la numérotation appliquée à l'arbre XML a été simplifiée. En fait, il est possible de sélectionner jusqu'à une lettre d'un nœud texte d'un élément. Pour plus d'information, consulter <http://www.w3.org/TR/xptr-xpointer/>
- Il existe aussi dans le framework XPointer l'équivalent de l'élément BASE du HTML avec l'attribut `xml:base` :

```
<?xml version="1.0"?>
<doc xml:base="http://example.org/today/" xmlns:xlink="http://www.w3.org/1999/xlink">
  <head>
    <title>Virtual Library</title>
  </head>
  <body>
    <paragraph>See <link xlink:type="simple" xlink:href="new.xml">what's new</link>!</paragraph>
  </body>
</doc>
```


Conclusion

- Le couple XLink/XPointer sera peut être le successeur de la balise <a> du HTML
- Il existe encore peu d'applications dans lesquelles XLink et XPointer sont implantés
 - Fujitsu XLink Processor est un rare exemple d'implantation de XLink et d'une partie de XPointer
- D'autres spécifications existent pour décrire des ressources (une ressource est un document situé à un URI particulier)
 - RDF (Resource Description Framework) est un langage XML permettant de décrire des ressources comme par exemple le contenu d'un livre
 - Le Dublin Core est un ensemble de standard de 15 informations précises dédiées à la gestion d'un catalogue ou d'une bibliothèque

- Durée prévue [Début – XLink/Xpointer] : 1h40 minutes. Durée prévue [XPath – XLink/Xpointer] : 10 minutes.
- Point à souligner : le couple XLink/Xpointer est un exemple d'application XML
- Un autre point de vue sur la manière de traiter les liens en XML est disponible à l'adresse <http://xmlfr.org/documentations/tutoriels/links/slide2>
- Fujitsu XLink Processor : <http://www.labs.fujitsu.com/free/xlip/en/index.html>
- Pour de l'information sur RDF, voir <http://www.w3.org/RDF/> : ce site utilise d'ailleurs un index constitué dynamiquement par chargement dans une base [RDF](#) de documents [RSS](#) (RDF Site Summary) décrivant le site
- Pour de l'information sur Dublin Core, voir <http://purl.org/dc/>

Manipulations de documents XML

Avril 2003

Direction des systèmes d'information

Introduction

Transformations de documents XML

Principes généraux

CSS

XSLT

XSL-FO

XPath

Liaisons de documents en XML

Principes généraux

XLink

XPointer

Manipulations de documents XML

Principes généraux

SAX

DOM

Conclusion

Références

- ❑ La problématique se situe autour de la manière de manipuler des documents XML depuis des programmes (lecture, écriture, modification)
 - On appelle souvent analyseur syntaxique un composant logiciel qui implante des fonctions de manipulations de documents XML
 - On dit qu'un programme « parse » un document XML lorsque le programme lit (voire valide) le document

- ❑ 2 APIs (Application Programming Interface) standardisent la manipulation de documents XML
 - SAX (Simple API for XML), API qui aborde la notion de parsing de manière événementielle
 - DOM (Document Object Model), API qui traite un document XML avec une approche arborescente

Introduction

Transformations de documents XML

Principes généraux

CSS

XSLT

XSL-FO

XPath

Liaisons de documents en XML

Principes généraux

XLink

XPointer

Manipulations de documents XML

Principes généraux

SAX

DOM

Conclusion

Références

□ Le parseur lit le document XML du début à la fin (séquentiellement) et génère un évènement à chaque information significative rencontrée (attribut, élément, texte, ...)

- Soit l'extrait de document XML suivant

```
<nom><prenom>David</prenom><nom_famille>Rousse</nom_famille></nom>
```

- Un parseur SAX reportera typiquement les événements suivants (dans l'ordre indiqué)

```
startElement : nom  
startElement : prenom  
content : David  
endElement : prenom  
startElement : nom_famille  
content : Rousse  
endElement : nom_famille  
endElement : nom
```

- Spécifications de la version 2.0.1 disponible à l'adresse <http://www.saxproject.org/>

-
- API spécifiée à l'origine pour le langage JAVA, devenue un **standard de fait** disponible dans plusieurs langages de programmation
 - La fondation Apache fournit une version du parseur Xerces qui implante l'API SAX en JAVA et une version du parseur Xerces qui implante l'API SAX en C++

 - Avantages
 - Ne nécessite pas un chargement complet en mémoire du document XML à parser
 - Adapté à des opérations de transformation de documents XML

 - Inconvénients
 - Non adapté aux opérations de parcours et d'écriture de documents XML

- Les outils actuels (Xerces par exemple) implantent les versions 1.0 et 2.0 de SAX.

Étapes nécessaires lors de l'écriture d'un programme JAVA implémentant l'API SAX

- Implanter l'interface ContentHandler
- Implanter l'interface ErrorHandler
- Instancier un parseur
- « Enregistrer » de l'instance de la classe implémentant ContentHandler dans le parseur
- « Enregistrer » de l'instance de la classe implémentant ErrorHandler dans le parseur
- Fixer les propriétés de validation du parseur
- Parser le document XML

- L'exemple se base sur une implémentation en JAVA de l'API SAX 2.0 avec Apache Xerces 2.3.0 (<http://xml.apache.org/>).
- L'interface ContentHandler contient la liste des méthodes (les callbacks dans la terminologie SAX) qui permettent de traiter le contenu du document XML.
- L'interface ErrorHandler contient la liste des méthodes dédiées aux éventuelles erreurs de parsing.

□ Implantation de l'interface ContentHandler

```
public class JSimpleContentHandler implements ContentHandler {  
    public void startDocument() throws SAXException {  
    }  
    public void endDocument() throws SAXException {  
    }  
    public void processingInstruction(String target, String data) throws SAXException {  
    }  
    public void startPrefixMapping(String prefix, String uri) {  
    }  
    public void endPrefixMapping(String prefix) {  
    }  
  
    public void startElement(String namespaceURI, String localName, String qName, Attributes atts) throws SAXException {  
        System.out.println("Element: " + localName);  
  
        for (int i=0; i<atts.getLength(); i++) {System.out.println("Attribut (nom = " +atts.getLocalName(i) + ", valeur = " +  
            atts.getValue(i) + ")");  
        }  
    }  
  
    public void endElement(String namespaceURI, String localName, String qName) throws SAXException {  
    }  
    public void characters(char[] ch, int start, int length) throws SAXException {  
    }  
    // ATTENTION, classe tronquée pour les besoins de présentation du diaporama ...  
}
```

- Les méthodes de ContentHandler montrent les « événements qui seront levés par le parseur SAX.

□ Implantation de l'interface ErrorHandler

```
public class JSimpleErrorHandler implements ErrorHandler {  
  
    public void warning(SAXParseException exception) throws SAXException {  
    }  
  
    public void error(SAXParseException exception) throws SAXException {  
    }  
  
    /**  
     * Reporte les erreurs fatales  
     */  
    public void fatalError(SAXParseException exception) throws SAXException {  
  
        System.out.println("**Erreur fatale de parsing**\n" +  
            " Ligne: " +  
                exception.getLineNumber() + "\n" +  
            " URI: " +  
                exception.getSystemId() + "\n" +  
            " Message: " +  
                exception.getMessage());  
        throw new SAXException("Erreur fatale rend=");  
    }  
}
```

❑ Instanciation d'un parseur

```
private String vendorParserClass = "org.apache.xerces.parsers.SAXParser";  
// création d'une instance du parser  
XMLReader reader = XMLReaderFactory.createXMLReader(vendorParserClass);
```

❑ « Enregistrement » de l'instance de la classe implémentant ContentHandler dans le parseur

```
ContentHandler myContentHandler = new JSimpleContentHandler();  
// enregistrement du content handler  
reader.setContentHandler(myContentHandler);
```

❑ « Enregistrement » de l'instance de la classe implémentant ErrorHandler dans le parseur

```
ErrorHandler myErrorHandler = new JSimpleErrorHandler();  
// enregistrement du error handler  
reader.setErrorHandler(myErrorHandler);
```

- On aurait pu instancier le reader avec la ligne de code suivante :
XMLReader reader = new org.xml.sax.SAXParser();

SAX – Exemple (5/5)

 Fixation des propriétés de validation du parseur

```
// validation
reader.setFeature("http://xml.org/sax/features/validation", true);
reader.setFeature("http://apache.org/xml/features/validation/schema-full-checking", true);
```

 Parsing du document XML

```
// parsing
InputSource inputSource = new InputSource(xmlURI);
reader.parse(inputSource);
```

Introduction

Transformations de documents XML

Principes généraux

CSS

XSLT

XSL-FO

XPath

Liaisons de documents en XML

Principes généraux

XLink

XPointer

Manipulations de documents XML

Principes généraux

SAX

DOM

Conclusion

Références

DOM – Définition

- API spécifiée en IDL par le W3C, par essence multi langages, proposant une vision en arbre du document XML à traiter
- Un composant logiciel implantant DOM permet ainsi de réaliser des traitements (parcours, modifications, ...) des nœuds de l'arbre représentant le document XML original
 - La fondation Apache fournit une version du parseur Xerces qui implante l'API DOM en JAVA et une version du parseur Xerces qui implante l'API DOM en C++
- Avantages
 - Adapté aux opérations de parcours et d'écriture (sérialisation) de documents XML
- Inconvénients
 - Nécessite un chargement complet en mémoire du document XML à parser

- Spécifications disponible à l'adresse <http://www.w3.org/DOM/>
- La version DOM Level 2 est actuellement la version du DOM implantée dans les outils (Xerces notamment).
- La spécification DOM Level 3 est au stade de Working Draft.
- IDL=Interface Definition Language, langage de définition d'interface, utilisé par exemple dans CORBA pour spécifier l'interface d'un composant.

DOM – Spécification IDL (extrait)

```
interface Node {
// déclaration de l'ensemble des const unsigned short supprimée pour les besoins de présentation du diaporama
readonly attribute DOMString      nodeName;
        attribute DOMString      nodeValue;
readonly attribute unsigned short  nodeType;
readonly attribute Node           parentNode;
readonly attribute NodeList       childNodes;
readonly attribute Node           firstChild;
readonly attribute Node           lastChild;
readonly attribute Node           previousSibling;
readonly attribute Node           nextSibling;
readonly attribute NamedNodeMap   attributes;
readonly attribute Document       ownerDocument;
Node      insertBefore(in Node newChild, in Node refChild) raises(DOMException);
Node      replaceChild(in Node newChild, in Node oldChild) raises(DOMException);
Node      removeChild(in Node oldChild) raises(DOMException);
Node      appendChild(in Node newChild) raises(DOMException);
boolean   hasChildNodes();
Node      cloneNode(in boolean deep);
void      normalize();
boolean   isSupported(in DOMString feature, in DOMString version);
readonly attribute DOMString      namespaceURI;
        attribute DOMString      prefix;
readonly attribute DOMString      localName;
boolean   hasAttributes();
};
```

- **L'interface Node est la racine de la hiérarchie de classes du DOM. Elle est la base de la vision arborescente d'un document XML fourni par le DOM.**

Étapes nécessaires lors de l'écriture d'un programme JAVA implantant l'API DOM

- Écrire le code de la classe *DOMSimple* qui utilise une instance de Document
- Instancier un parseur
- Parser le document XML
- Instancier la classe *DOMSimple*
- Appeler la méthode de sérialisation de l'instance de *DOMSimple*

- L'exemple se base sur un implantation en JAVA de l'API DOM 2.0 avec Apache Xerces 2.3.0 (<http://xml.apache.org/>).
- L'exemple propose de lire un document XML source et de l'afficher tel quel à l'écran.
- L'étape de fixation des propriétés de validation du parseur est également à écrire, si besoin.

□ Code de la classe *DOMSimple* qui utilise une instance de Document

```
public class DOMSimple {  
  
    //ATTENTION, lignes de codes supprimées pour les besoins du diaporama ...  
  
    public void serializeNode(Node node, String indentLevel) throws IOException {  
  
        switch (node.getNodeType()) {  
  
            case Node.DOCUMENT_NODE:  
                System.out.println("<?xml version='1.0' encoding='ISO-8859-1'?>");  
                System.out.println(lineSeparator);  
  
                // recursion sur chaque enfant  
                NodeList nodes = node.getChildNodes();  
                if (nodes != null) {  
                    for (int i=0; i<nodes.getLength(); i++) {  
                        serializeNode(nodes.item(i), "");  
                    }  
                }  
                break;  
  
            case Node.TEXT_NODE:  
                System.out.println(node.getNodeValue());  
                break;  
        }  
  
        //ATTENTION, lignes de codes supprimées pour les besoins du diaporama ...  
    }  
}
```

Instanciation d'un parseur

```
DOMParser parser = new org.apache.xerces.parsers.DOMParser();
```

 Parsing du document XML

```
parser.parse(xmlDocument);  
Document doc = parser.getDocument();
```

 Instanciation de la classe *DOMSimple*

```
DOMSimple serializer = new DOMSimple();
```

 Appel de la méthode de sérialisation de l'instance de *DOMSimple*

```
serializer.serialize(doc);
```

- **parser.parse(xmlDocument);** renvoie void comme avec SAX et il est donc nécessaire d'écrire **Document doc = parser.getDocument();** pour récupérer l'arbre DOM.

Conclusion

- SAX est bien adapté à des opérations de transformations (outils d'EAI par exemple)
- DOM est bien adapté pour le parcours et la sérialisation (outils d'édition de documents par exemple)
- Il existe d'autres manières de manipuler du XML via programme
 - JDOM, API restreinte au monde JAVA fournissant de fonctionnalités comparables à DOM mais spécifiée dans l'esprit orienté objet
 - JAXP (Java API for XML Processing), couche d'abstraction proposée par Sun au dessus des APIs XML telles que SAX, DOM et JDOM pour normaliser l'écriture de programmes JAVA manipulant des documents XML
 - Data Binding, technique permettant de générer automatiquement des classes JAVA à partir d'un document XML (la classe JAVA est vue comme la DTD/le schéma XML et les instance de la classe JAVA comme des documents XML se conformant à la DTD/le schéma XML). Des outils comme JAXB (JAVA Architecture for XML Binding), Castor et Zeus fournissent ce service

- Et l'on aurait aussi pu parler de XML-RPC, de SOAP et des Web Services ...

Introduction

Transformations de documents XML

Principes généraux

CSS

XSLT

XSL-FO

XPath

Liaisons de documents en XML

Principes généraux

XLink

XPointer

Manipulations de documents XML

Principes généraux

SAX

DOM

Conclusion

Références

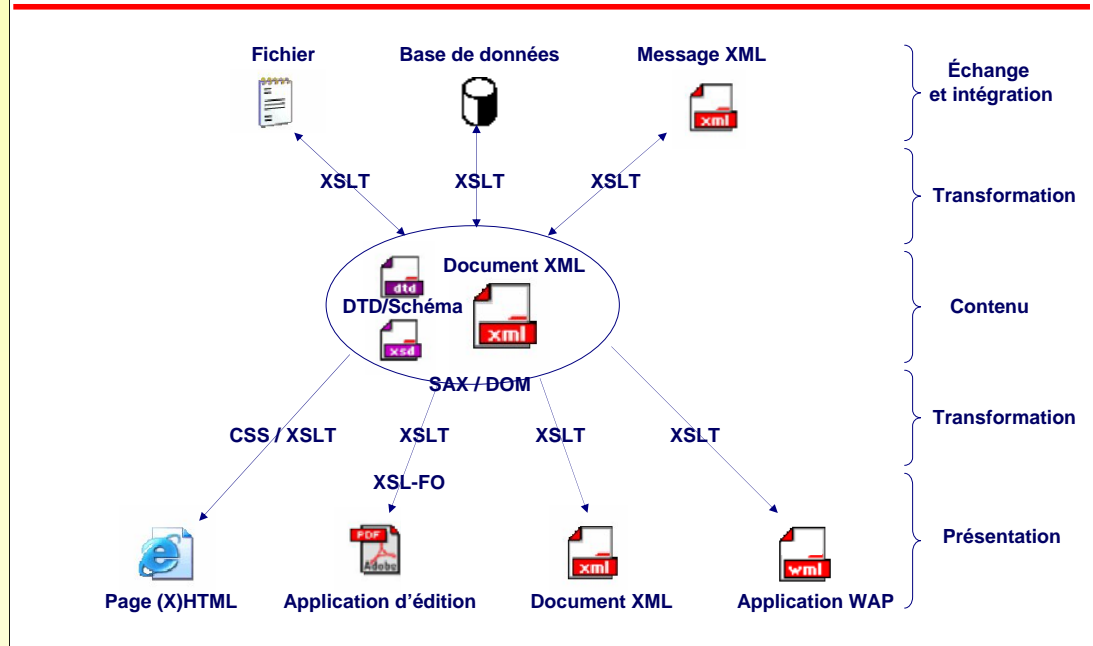
Conclusion

Avril 2003

Direction des systèmes d'information

- Le document XML contient les données : la notion de document bien formé intervient à ce niveau pour vérifier que le contenu du document respecte la syntaxe XML
- Les DTD ou les schémas XML permettent de décrire des contraintes sur ce que peut contenir un document XML : la notion de document valide intervient à ce niveau
- Les langages de présentation ont pour objet de présenter les données contenues dans un document XML : présentées à l'utilisateur en HTML via CSS, destinées à l'impression en PDF via XSL-FO
- Le langage de transformation XSLT joue un rôle central pour transformer un document (ou une partie d'un document) XML en un « autre document »
- XLink et XPointer représentent deux exemples d'applications XML permettant d'exprimer des liens avec une syntaxe XML
- Les manipulations de documents XML depuis un langage de programmation passent par l'utilisation d'APIs normalisées comme SAX et DOM

- **Durée prévue [Début – Conclusion] : 2h. Durée prévue [XLink/XPointer – Conclusion] : 20 minutes.**



- Les points essentiels à voir :
- XML permet de décrire l'information indépendamment des applications (comme son ancêtre SGML).
- XML s'occupe du contenu, XSLT des transformations, CSS/XSL-FO, ... de la présentation (séparation des couches données/traitement/présentation).
- Donc XML permet de partager (entre applications, pour faire de l'EAI, ...) et réutiliser l'information autour de l'Internet.
- Remarque : ne sont pas indiqués dans ce schéma XPath et le couple XLink/XPointer vus pendant la formation.
- Pour aller plus loin, une liste (non exhaustive) d'ouvrages traitant d'XML est présentée dans le transparent suivant.

Livres

- XML in a nutshell (2° édition), Eliotte Rusty Harold et W. Scott Means, O'Reilly
- XSLT, Doug Tidwell, O'Reilly
- Comprendre XSLT, Bernd Amann et Philippe Rigaux, O'Reilly
- Java et XML (2° édition), Brett McLaughlin, O'Reilly

Sites Web

- Le World Wide Web Consortium, www.w3.org
- Le portail <XML> fr, xmlfr.org
- La fondation Apache, xml.apache.org