XML

eXtensible Markup Language

Reda Bendraou
Reda.Bendraou@Lip6.fr

Cours en Anglais:

http://pagesperso-systeme.lip6.fr/Reda.Bendraou/Enseignements.htm

Plan

- **Partie I :** Le standard XML
  - Objectifs
  - Pourquoi XML ?
  - Structure d'un document XML
  - Document bien formé

- Partie II : Definition des documents XML
  - DTD
  - XML Schema

- Partie III : Mise en forme, Traitement et Transformations des documents XML
  - Mise en forme : les feuilles de style XSL
  - DOM *(Documment Object Model)*
  - XPath *(Chemins d'accès au arbre XML)*
  - Transformations XSLT

Objectifs

- On veut représenter des données
  - Facilement **lisibles** : - par les **humains**
    - par les **machines**
  - Selon une technologie **compatible WEB**
    (à intégrer facilement dans les serveurs WEB)
  - **en séparant les aspects** : - **présentation** *(format, couleurs etc..)*
    - **information** *(données)*
  - D'une manière **standardisée**

Pourquoi XML ?
(Etat de l'art)

Formats existants :

- **HTML** = **HyperText** Markup Language
- **SGML** = **Standard Generalized** Markup Language

→ *Langage à balises*

Autres notations :

- **ASN.1=** Abstract Syntax Notation (ITU-T)
- CDR, XDR = Common/eXtenal Data Representation
- etc…..

Critique de HTML

- Langage **simple**, **lisible !** *(texte formaté)*
- **Compatible WEB !**
- **Non extensible !** **(***Nombre fixe de balises et attributs)*
- **Mélange des genres !**

  (i.e. balise de structuration et de mise en forme : <H1> title 1 </H1>*)*
- **Incompatibilité** entre navigateurs et versions !
- **Pas de preuve** sur le document
  - structure *(ordre des balises)*,
  - données *(type, valeur)*,
  - sémantique

Critique de SGML

- Langage **puissant**, **extensible, standard (**ISO 8879-1986**)!**
- **Méta-langage** de documentation pour grosses applications

(i.e. automobile, avion, dictionnaire, etc…)

…*mais*

- **Trop complexe** ! -> Implémentation beaucoup trop lourde !
- **Pas forcement compatible WEB** !

XML

Définition intuitive d'XML:

- **XML** :
  - → - variante de **HTML généralisé** !
    (compatibilité WEB, lisibilité, syntaxe)
  - → - **sous-ensemble de SGML** !
    (flexibilité, rigueur)

- **langage à balises configurables**
- pour la représentation hiérarchique de données,
- http://www.w3.org/XML/

Structure de documents XML

- **Prologue :**
  - rôle équivalent au <HEAD> HTML,
  - Meta–Informations :
    - **instructions** de traitement
    - **commentaires**

    (non interprétables par le parseur)
- **Corps :**
  - rôle équivalent au <BODY> HTML
  - les donnes formatées :
    - **balises** d'encadrement
    - **attributs associées** aux balises
    - **données encadrées** par les balises

    (structure arborescente)

Exemple XML : *Une lettre*

PROLOGUE

```
<?xml version = "1.0" standalone="yes" encoding="ISO8859-1"?>
```

document XML instruction de traitement

document autonome

jeu de caractères utilisé (latin)

balise début

CORPS

```
<lettre>
    <lieu> Somewhere in space</lieu>
    <expediteur> ObiWan Kenobi </expediteur>
    <destinataire> Luke Skywalker </destinataire>
    <introduction> Cher padawan, </introduction>
    <corps_lettre> …May the force be with you </corps_lettre>
    <signature/>
</lettre>
```

données balisées

balise unique (sans données)

balise fin

Prologue d'un document **XML**

(Exemple)

ceci est un document XML non autonome
(il utilise une définition externe)

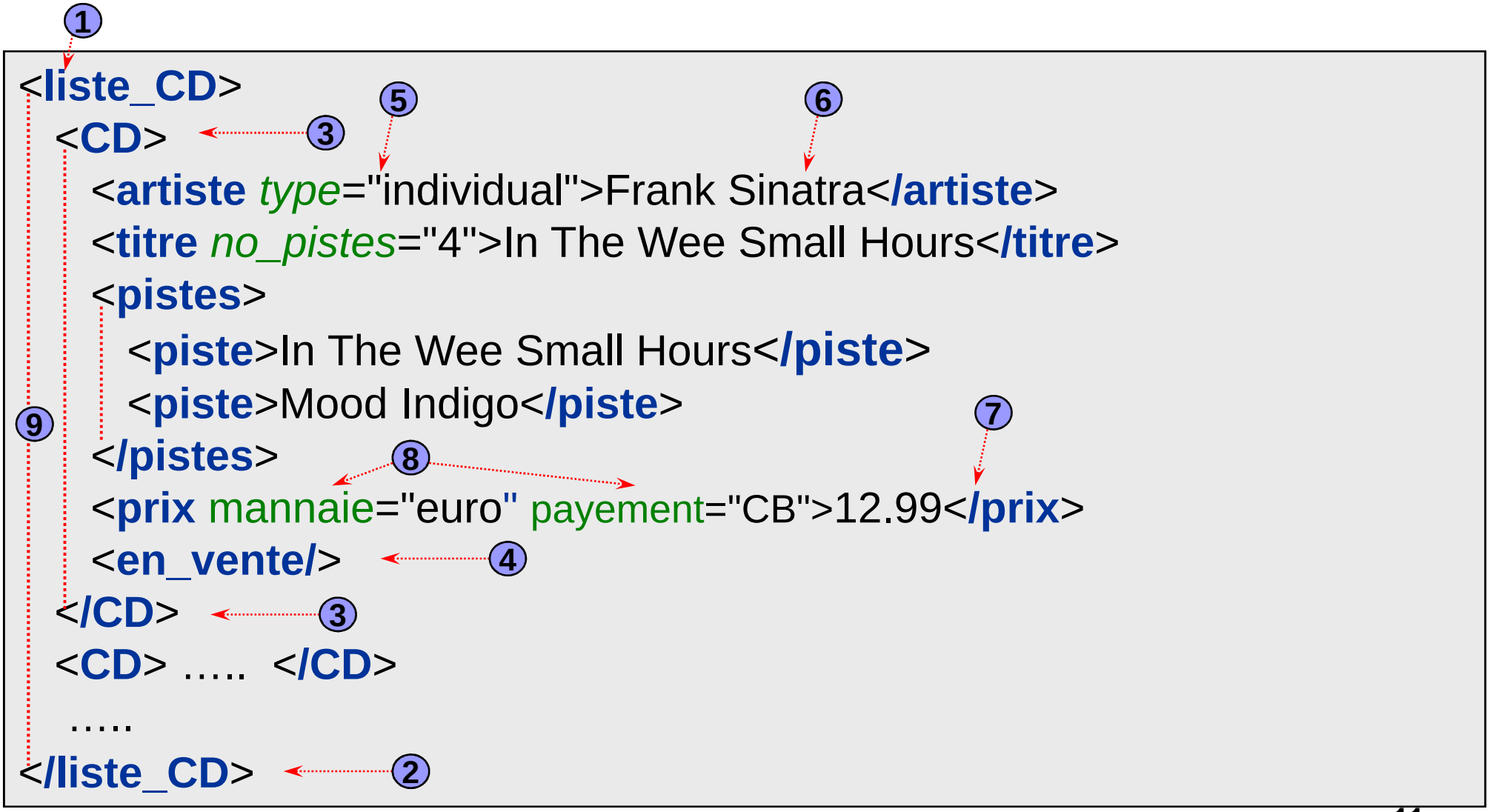
document XML 1.0

```
<?xml version="1.0" standalone = "no"  encoding="ISO8859-1" ?>
<!DOCTYPE liste_CD SYSTEM "CDs.dtd">
```

un commentaire spécial !
(il définit le type de document XML)

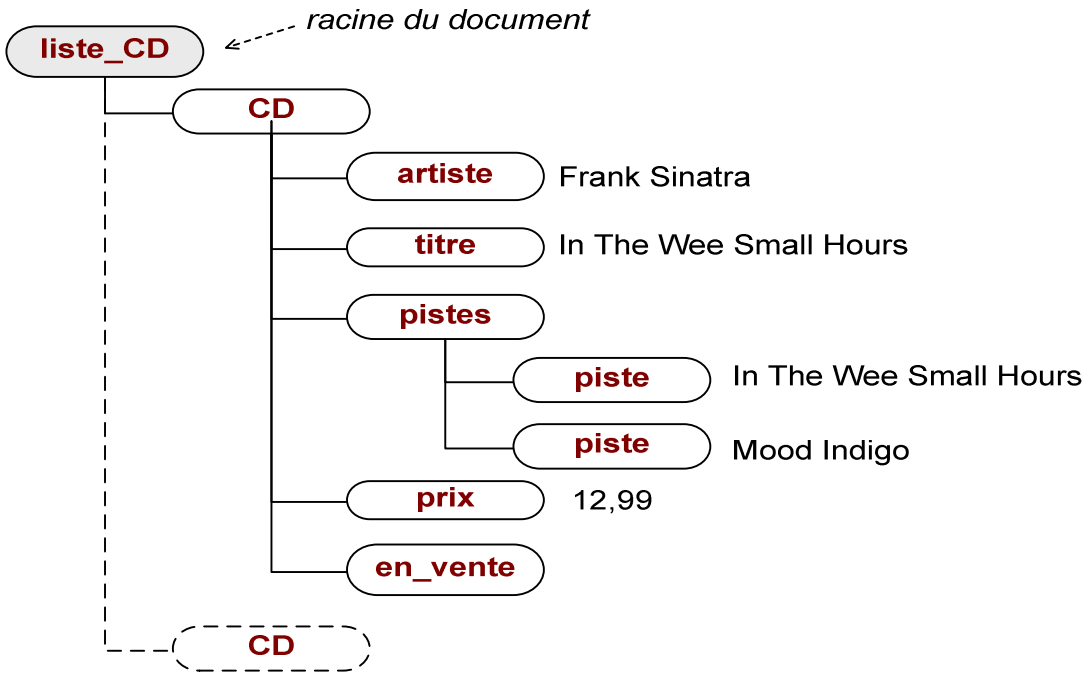
conforme à une définition externe
(spécifié dans le fichier "CDs.dtd")

Corps d'un document **XML** (**Exemple**)



Corps d'un document XML

(arbre des balises sur l'exemple)



Corps d'un document XML

(explications sur l'exemple)

- Balisage arborescent *(voir le transparent 12)*
- La **racine** du corps est **unique (1)(2)**.
- Les balises sont soit :
  - par **paires** : début **(1)** ,et fin **(2),**
  - **uniques (4)**.
- Le contenu entre deux balises paires **(3)** est soit :
  - **une valeur simple** : *chaîne de caractère* **(6)**, *numéro réel* **(7),** etc.,
  - **une arborescence d'autres balises (9)**.
  - **un mélange des deux** *(pas présent dans l'exemple).*
- Certaines balises (de début) content des **attributs (5)(8)**,

Structure des documents XML : Synthèse

- Un document XML : Prologue + Corps

(un arbre de balises)

- Balises du prologue :

instruction de traitement

```
<?nom_balise_traitement  ....  ?>
```

➢*transmis directement à une application spécifique*

commentaire

```
<!DOCTYPE .....>
```

- Balises du corps par paires (conteneurs pour les données) ou uniques

```
<nom_balise nom_attribut1= "val" nom_attribut2="val"> contenu
</nom_balise>
```

```
<nom_balise_simple/>
```

Attributs d'une balise XML : Compléments

- Attributs XML = **Données cachées** non visualisées par un navigateur (sauf si explicitement demandé)
- **Structuration "à plat"** !
- **Pas d'ordre** de précédence !
- Syntaxe : **nom='valeur'** *ou* **nom="valeur"**
- Caractères interdits : **^**, **%** et **&**
- Attributs prédéfinis :
  - xml:lang="fr"
  - xml:id="identificateur_unique_de_la_balise"
  - xml:idref ="reference_vers_une_balise"

exemple :

```
<livre langue="FR" date_debut="09/2000" id="ISBN-123"/>
```

Un Document XML **bien formé**

- **Un document XML avec une syntaxe correcte est dit *bien formé* =>**
  - Le document XML doit avoir un seul élément racine
  - Les éléments (balises) XML doivent avoir une balise fermente
  - Les balises XML sont sensibles à la casse (case-sensitive)
  - Les valeurs des attributs doivent toujours être entre guillemets
  - Les balises XML ne doivent pas se chevaucher

Document XML **bien formé**

- Conforme aux **règles syntaxiques** du langage XML !

contre exemple :

balises chevauches

```
<?xml version = "1.0" standalone="yes"?>
<!-- Document XML pas bien formé ! -->
<peintre>
  <nom> Picasso
    <prenom>
  </nom> Pablo
    </prenom>
</peintre>
```

- Alors :
  - Association possible avec une feuille de style
  - Peut être exploité par un parseur/analyseur syntaxique
    (i.e. pour parcourir l'arbre XML et le transformer)
  - Candidat pour être valide

Document XML **valide**

- Associé à une définition **DTD** (.dtd) ou un **Schema** (.xsd)
  - définition :
    - **-interne** au document XML → **non recommandé**
      (dans le commentaire DOCTYPE)
    - **-externe** → **réutilisation des définitions**, **échange**
      (référencé vers un fichier dans le DOCTYPE)
- **Conditions :**
  - document bien formé *(syntaxe correcte),*
  - structure du document respectant la définition (voir les DTD),
  - les références aux éléments du document soit résolues
- Alors
  - Le document XML peut être échangé ! *(format standardisé)*

Exercice

- Proposez un document XML bien formé représentant un ensemble de références bibliographiques
- Si vous vouliez traiter ces références bibliographiques, de quoi aurez vous besoin ?
- Quelles sont vos premières réflexions sur XML ?

Premières réflexions XML (1)

Ce document ne spécifie pas :

- pour les **balises** :
  - le **noms** des balises
  - **contraintes** sur :
    - **l'ordre**,
    - **la multiplicité** *(no. occurrences)*,
    - **la composition** *(la hiérarchie).*
- pour les **attributs** :
  - le **nom** des attributs *(pour chaque balise)*
  - le **type** des attributs *(i.e. chaîne, énumération, etc.)*
  - les **valeurs** des attributs *(i.e domaine, format etc.)*
  - **contraintes** sur leur valeurs *(i.e format, domaine etc)*
- pour le **contenu** de balises : - le **type** des données *(i.e. chaîne de caractères, énumération, etc.)*

Premières réflexions XML (2)

- Questions :
  - Quand utilise-t-on des balises et quand utilise-t-on des attributs?
    - **balises** → entités
    - **attributs** → propriétés
  - Comment spécifie-t-on ce qui doit être affiché et comment ?
    - **style** → CSS
    - **transformations** → XSLT, DOM, XPath …..
  - L'ordre des attributs est-t-il important ?
    - → non

XML
(*Partie II*)

Definition des documents XML

DTD, XML Schema

Document bien formé et valide

- Document bien formé
  - Respecte les règles d'écriture syntaxique
  - pas nécessairement conforme à une DTD ou XML schema
- Document valide
  - bien formé + conforme à une DTD (ou un schéma)

DTD

- Permet de définir le «vocabulaire» et la structure qui seront utilisés dans le document XML
- Grammaire du langage dont les phrases sont des documents XML (instances)
- Peut être mise dans un fichier et être référencé dans le document XML

Elément et attribut

- <!ELEMENT *balise* (contenu)>
  - Décrit une *balise* qui fera partie du vocabulaire.

Syntax:

  - <!ELEMENT tag (content) > Ou bien
  - <!ELEMENT element-name category> (i.e. EMPTY, ANY, #PCDATA)
    - Exp. : <!ELEMENT book (author, editor)>

- <!ATTLIST *balise* [attribut type #mode [valeur]]*
  - Définit la liste d'attributs pour une balise
  - ex : <!ATTLIST auteur
    genre CDATA #REQUIRED
    ville CDATA #IMPLIED>
    <!ATTLIST editeur
    ville CDATA #FIXED "Paris">

Structuration des balises

- Structuration du contenu d'une balise
  - (a, b) séquence *ex* (nom, prenom, rue, ville)
  - (a|b) liste de choix *ex* (oui|non)
  - a? élément optionnel [0,1] *ex* (nom, prenom?, rue, ville)
  - a* élément répétitif [0,N] *ex* (produit*, client)
  - a+ élément répétitif [1,N] *ex* (produit*, vendeur+)

Types de données

- CDATA
  - Données brutes qui ne seront pas analysées par le parseur
- PCDATA
  - Elément de texte sans descendants ni attributs contenant des caractères
- Enumération
  - Liste de valeurs séparées par « | »
- ID et IDREF
  - Clé et référence pour les attributs
- ANY
  - Tout texte possible - pour le développement
- EMPTY
  - Vide

Exemple de DTD Externe *(fichier .dtd)*

```
<!ELEMENT doc (livre* | article+)>
<!ELEMENT livre (titre, auteur+)>
<!ELEMENT article (titre, auteur*)>
<!ELEMENT titre(#PCDATA)>
<!ELEMENT auteur(nom, adresse)>
        <!ATTLIST auteur id ID #REQUIRED>
<!ELEMENT nom(prenom?, nomfamille)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT nomfamille (#PCDATA)>
<!ELEMENT adresse ANY>
```

Exemple de DTD interne

```
<?XML version="1.0" standalone="yes"?>

<!DOCTYPE CATALOGUE [
      <!ELEMENT CATALOGUE (VOITURES +)>

      <!ELEMENT VOITURES (SPECIFICATION+, ANNEE, PRIX)>
      <!ATTLIST VOITURES NOM CDATA #REQUIRED>

      <!ELEMENT SPECIFICATION EMPTY>
      <!ATTLIST SPECIFICATION  MARQUE CDATA #REQUIRED
                               COULEUR CDATA #REQUIRED>

      <!ELEMENT ANNEE (#PCDATA)>
      <!ELEMENT PRIX (#PCDATA)>
  ]>

<CATALOGUE>
  <VOITURES NOM= " LAGUNA">
    <SPECIFICATION MARQUE= " RENAULT" COULEUR="Rouge"/>
    <ANNEE>2001</ANNEE>
    <PRIX>10 000 Euros</PRIX>
  </VOITURES>
  ……..
</CATALOGUE>
```

Exemple de ID et IDREF

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE DOCUMENT [
    <!ELEMENT DOCUMENT(PERSONNE*)>
    <!ELEMENT PERSONNE (#PCDATA)>
    <!ATTLIST PERSONNE PNUM ID #REQUIRED>
    <!ATTLIST PERSONNE MERE IDREF #IMPLIED>
    <!ATTLIST PERSONNE PERE  IDREF #IMPLIED>
]>
<DOCUMENT>
    <PERSONNE PNUM = "P1">Marie</PERSONNE>
    <PERSONNE PNUM = "P2">Jean</PERSONNE>
    <PERSONNE PNUM = "P3" MERE="P1" PERE="P2">Pierre</PERSONNE>
    <PERSONNE PNUM = "P4" MERE="P1" PERE="P2">Julie</PERSONNE>
</DOCUMENT>
```

Pourquoi des DTD externes ?

- Modèle pour plusieurs documents
  - partage des balises et structures
- Définition locale ou externe
  - <!DOCTYPE doc SYSTEM "doc.dtd">
  - <!DOCTYPE doc PUBLIC "www.e-xmlmedia.com/doc.dtd">
- Exemple de document

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE VOITURES SYSTEM "voitures.dtd">
...
```

Entités dans les DTD

- Entité
  - Permet la réutilisation dans une DTD

  Syntax
  - Déclaration interne:
    <!ENTITY entity-name entity-value>
  - Déclaration externe:
    <!ENTITY entity-name SYSTEM "entity-URL">
  - Pour la référencer → **&**entity-name
  - Exemple (interne):
    <!ENTITY website "http://www.TheScarms.com">

    Exemple (externe):
    <!ENTITY website SYSTEM "http://www.TheScarms.com/entity.xml">

- Dans un document XML:
  <url>&website</url>

  Sera évaluée à:
  <url>http://www.TheScarms.com</url>

Synthèse DTD

- Spécification de la structure du document
  - déclaration de balisage : ELEMENT, ATTLIST, ENTITY;
  - déclaration des éléments
    - éléments simples :
      - **vide** (EMPTY)
      - **libre** (ANY),
      - **textuel** (#PCDATA)
    - composition :
      - **séquence d'éléments** liste ordonnée → (a,b,c)
      - **choix alternatives** d'éléments → (a|b|c)
      - **mixte hiérarchique** → (a, (b|c),d)
    - indicateurs d'occurrences :
      - **?** (zéro ou une)),
      - ***** (zero ou plusieurs),
      - **+** (une ou plusieurs)

Exercice

- Proposez une DTD permettant de définir des document XML représentant des références bibliographiques.
- Si vous voulez extraire des données de ces documents XML selon votre DTD, quelles sont les difficultés qui pourraient être posées.
- Quelles sont les avantages et les inconvénients des DTDs.

Insuffisance des DTD

- Pas de types de données
  - difficile à interpréter
  - difficile à traduire en schéma objets
  - Pas d'héritage
- Propositions de compléments
  - XML-schema du W3C

Objectifs des schémas

- Reprendre les acquis des DTD
  - plus riche et complet que les DTD
- Permettre de typer les données
  - éléments simples et complexes
  - attributs simples
- Permettre de définir des contraintes
  - occurrence obligatoire ou optionnelle
  - cardinalités, références
- Réutilisation avec les espaces de nommages

XML Schéma

- Un schéma d'un document XML définit
  - les éléments possibles dans le document
  - les attributs associés à ces éléments
  - la structure du document et **les types de données**
- Le schéma est spécifié en XML
  - pas de nouveau langage
  - balisage de déclaration
  - espace de nommage
- Présente de nombreux avantages
  - structures de données avec types de données
  - extensibilité par héritage
  - analysable par un parseur XML standard

Définir un schéma XML

- Document XML .xsd
- <schema> est l'élément racine

```
<?xml version="1.0"?>
<xsd:schema>
        //corps du schema..
        //...
 </xsd:schema>
```

- <schema> peut contenir certains attributs. La déclaration d'un schema est souvent comme suit :

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    //...
    //...
</xs:schema>
```

Référencer un schéma XML

- Ajouter la référence au niveau de la **balise racine** du document XML :

```
<?xml version="1.0"?>
<note xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="chemin_fichier.xsd">
        <to>Paul</to>
        <from>Alexis</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
</note>
```

Déclaration d'un élément simple

- Un élément simple contient des données dont le type est simple (ex: types de base en java)
  - **Ne contient pas d'autres éléments ni d'attributs**
- Un élément simple est défini selon la syntaxe suivante :

```
<xsd:element name = "….." type= "….." />
```

Exemple en schéma XML:

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

Correspondance en Document XML

```
<Department >13</Department>
```

- Et aussi

<xsd:element name="color" type="xsd:string" default="red"/> (valeur par défaut)

<xsd:element name="color" type="xsd:string" fixed="red"/> (valeur inchangeable)

Les types simples (1)

Type	*Description*
string	*représente une chaîne de caractères.*
boolean	*représente une valeur booléenne true ou false.*
decimal	*représente un nombre décimal*
float	*représente un nombre à virgule flottante.*
double	*représente un nombre réel double.*
duration	*représente une durée*
dateTime	*représente une valeur date/heure.*
time	*représente une valeur horaire (format : hh:mm:ss.sss).*
date	*représente une date (format : CCYY-MM-DD).*
gYearMonth	*représente un mois et une année grégorienne (format : CCYY-MM)*

Les types simples (2)

Type	*Description*
gYear	*représente une année (format : CCYY).*
gMonthDay	*représente le jour d'un mois (format : MM-DD)*
gDay	*représente le jour d'un mois (format : DD).*
gMonth	*représente le mois (format : MM).*
hexBinary	*représente un contenu binaire hexadécimal.*
base64Binary	*représente un contenu binaire de base 64.*
anyURI	*représente une adresse URI (ex.: http://www.site.com).*
QName	*représente un nom qualifié.*
NOTATION	*représente un nom qualifié.*

Les types simples (3)

Type	*Description*
Token	*représente une chaîne de caractères sans espaces blancs*
Language	*représente un langage exprimé sous forme de mot clés*
NMTOKEN	*représente le type d'attribut NMTOKEN (alphanumérique et . : - _)*
NMTOKENS	*représente le type d'attributs NMTOKEN + espace*
ID	*représente le type d'attribut ID*
IDREF, IDREFS	*représente le type d'attribut IDREF, IDREFS*
ENTITY, ENTITIES	*représente le type ENTITY, ENTITIES*
Integer	*représente un nombre entier*
nonPositiveInteger	*représente un nombre entier négatif incluant le zéro*
negativeInteger	*représente un nombre entier négatif dont la valeur maximum est -1*

Les types simples (4)

Type	*Description*
long	*représente un nombre entier long dont l'intervalle est : {-9223372036854775808 - 9223372036854775807}*
int	*représente un nombre entier dont l'intervalle est : {-2147483648 - 2147483647}*
short	*représente un nombre entier court dont l'intervalle est {-32768 - 32767}*
byte	*représente un entier dont l'intervalle est {-128 - 127}*
nonNegativeInteger	*représente un nombre entier positif incluant le zéro*
unsignedLong	*représente un nombre entier*
long	*non-signé dont l'intervalle est {0 - 18446744073709551615}*
unsignedInt	*représente un nombre entier non-signé dont l'intervalle est : {0 - 4294967295}*
unsignedShort	*représente un nombre entier court non-signé dont l'intervalle est : {0 - 65535}*
unsignedByte	*représente un nombre entier non-signé dont l'intervalle est {0 - 255}*
positiveInteger	*représente un nombre entier positif commençant à 1*

Déclaration d'un attribut

- Tous les attributs sont de type simple
  - **Syntaxe:**

    <xs:attribute name="xxx" type="yyy"/>

    Exp.

    <xs:attribute name="language" type="xs:string"/>

  - Aussi:

<xs:attribute name="lang" type="xs:string" default="EN"/> (si pas de valeur)

<xs:attribute name="lang" type="xs:string" fixed="EN"/> (ne peut être modifié)

  - Les attributs sont optionnels par default. Pour les rendre obligatoire, utiliser la propriété "use":

    <xs:attribute name="lang" type="xs:string" use="required"/>

Eléments Complexes

- Un élément complexe contient d'autres éléments et/ou attributs

- 4 types d'éléments complexes:
  - élément vide
  - élément qui contient d'autres éléments
  - élément qui contient que du texte et des attributs
  - élément qui contient du texte et d'autres éléments

- **Note:** chacun de ces éléments peut contenir des attributs en plus!

Les types complexes

- Déclarer un élément complexe = définir son type + association du type à l'élément
- Deux façon de déclarer un élément complexe
  1. Inclure la définition du type dans la déclaration de l'élément

Document XML

```
<employee>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
</employee>
```

Schéma XML correspondant :

```
<xsd:element name="employee">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="firstname" type="xsd:string"/>
            <xsd:element name="lastname" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Les types complexes

2. Exclure la définition du type de la déclaration de l'élément

Document XML

```
<employee>
        <firstname>John</firstname>
        <lastname>Smith</lastname>
</employee>
```

Schéma XML correspondant :

```
<xsd:element name="employee" type="personinfo"/>
//……
<xsd:complexType name="personinfo">
    <xsd:sequence>
        <xsd:element name="firstname" type="xsd:string"/>
        <xs:element name="lastname" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```

La seconde déclaration permet la réutilisation de types

Exemple : <xsd:element name="employee" type="personinfo"/>
<xsd:element name="student" type="personinfo"/>

Élément complexe, How to?

- **Définir des Complex Text-Only Elements**
  - Contenu simple (texte et attributs), ➔ ***simpleContent***
  - Exp.
    - **<shoesize country="france">35</shoesize>**

L'XML Schema correspondant:

```
<xs:element name="shoesize">
   <xs:complexType>
      <xs:simpleContent>   // to indicate that it does not contain other element
         <xs:extension base="xs:integer"> // to indicate the text type
            <xs:attribute name="country" type="xs:string" /> // the attribute type
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>
```

Élément complexe, How to?

- **Définir des Complex Types with Mixed Content**

  - Un élément complexe qui contient des **attributs**, des **éléments**, du **text**.
  - Exp.

<letter> Dear Mr.
<name>John Smith**</name>.** Your order **<orderid>**1032**</orderid>** will be shipped on **<shipdate>**2001-07-13**</shipdate>**
</letter>

L'XML Schema correspondant:

```
<xs:element name="letter">
    <xs:complexType mixed="true">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="orderid" type="xs:positiveInteger"/>
            <xs:element name="shipdate" type="xs:date"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Le type *sequence*

- Spécifie que les éléments fils doivent apparaître dans un ordre spécifique

Exemple

```
<xsd:element name="adresse">
   <xsd:complexType>
     <xsd:sequence>
              <xsd:element name="name" type="xsd:string"/>
              <xsd:element name="street" type="xsd:string"/>
              <xsd:element name="city" type="xsd:string"/>
              <xsd:element name="state" type="xsd:string"/>
              <xsd:element name="zip" type="xsd:decimal"/>
     </xsd:sequence>
     <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="FR"/>
   </xsd:complexType>
</xsd:element>
```

Exemple d'attribut déclaré dans un type

Le type *all*

- Spécifie que les éléments peuvent apparaître dans quelconque ordre
- Chaque élément fils doit apparaître une seule fois

Exemple

```
<xsd:element name="address">
   <xsd:complexType>
     <xsd:all>
            <xsd:element name= " firstname" type="xsd:string"/>
            <xsd:element name= " lastname" type="xsd:string"/>
     </xsd:all>
   </xsd:complexType>
</xsd:element>
```

Le type *Choice*

- Spécifie que seul un élément fils doit apparaître

Exemple

```
<xsd:element name="person">
   <xsd:complexType>
     <xsd:choice>
              <xsd:element name= " employee" type= "employee"/>
              <xsd:element name= " member" type= "member"/>
     </xsd:choice>
   </xsd:complexType>
</xsd:element>
```

D'autres types complexes

Indication d’occurrences

- Spécifie le nombre d’occurrence d’un élément
  - **maxOccurs**: le nombre maximum d’occurrence
  - **minOccurs** : le nombre minimum d’occurrence

Exemple

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
        <xs:element name="full_name" type="xs:string"/>
        <xs:element name="child_name" type="xs:string"
          maxOccurs="10"  minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Héritage de types

- Définition de sous-types par héritage de types simple ou complexes
  - Par extension : ajout d'informations
  - Par restriction : ajout de contraintes
- **Par extension :**

```
<xsd:complexType name="Address">
  <xsd:complexContent>
    <xsd:extension base="AddressFR">
      <xsd:sequence>
        <xsd:element name="pays"  type="xsd:string"/>
      </xsd:sequence>
    </xsd:/extension>
  </xsd:complexContent>
</xsd:complexType>
```

```
<xsd:complexType name="AddressFR>
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="street" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
      <xsd:element name="state" type="xsd:string"/>
      <xsd:element name="zip" type="xsd:decimal"/>
    </xsd:sequence>
</xsd:complexType>
```

Héritage de types (simple)

- **Par restriction :** en utilisant des expressions régulières (patterns), définir des contraintes sur des types simples

Exemple 1

```
<xsd:simpleType name="SKU">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="\d{3}-[A-Z]{2}"/>
        </xsd:restriction>
</xsd:simpleType>
```

Héritage de types (simple)

Exemple 2

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XML Schema Restrictions

Restrictions pour les types de données

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

XML Schema : exemple (1)

```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">

<xsd:element name="purchaseOrder" type="PurchaseOrderType"/>
<xsd:element name="comment" type="xsd:string"/>
<xsd:complexType name="PurchaseOrderType">
    <xsd:sequence>
        <xsd:element name="shipTo" type="USAddress"/>
        <xsd:element name="billTo" type="USAddress"/>
        <xsd:element ref="comment" minOccurs="0"/>
        <xsd:element name="items" type="Items"/>
    </xsd:sequence>
    <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```

Une référence

XML Schema : exemple (2)

```
<xsd:complexType name="Items">
    <xsd:sequence>
        <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="productName" type="xsd:string"/>
                    <xsd:element name="quantity">
                        <xsd:simpleType>
                            <xsd:restriction base="xsd:positiveInteger">
                              <xsd:maxExclusive value="100"/>
                            </xsd:restriction>
                        </xsd:simpleType>
                    </xsd:element>
                    <xsd:element name="USPrice" type="xsd:decimal"/>
                    <xsd:element ref="comment" minOccurs="0"/>
                    <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
                </xsd:sequence>
                <xsd:attribute name="partNum" type="SKU" use="required"/>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
 </xsd:schema>
```

Diagramme XML Spy

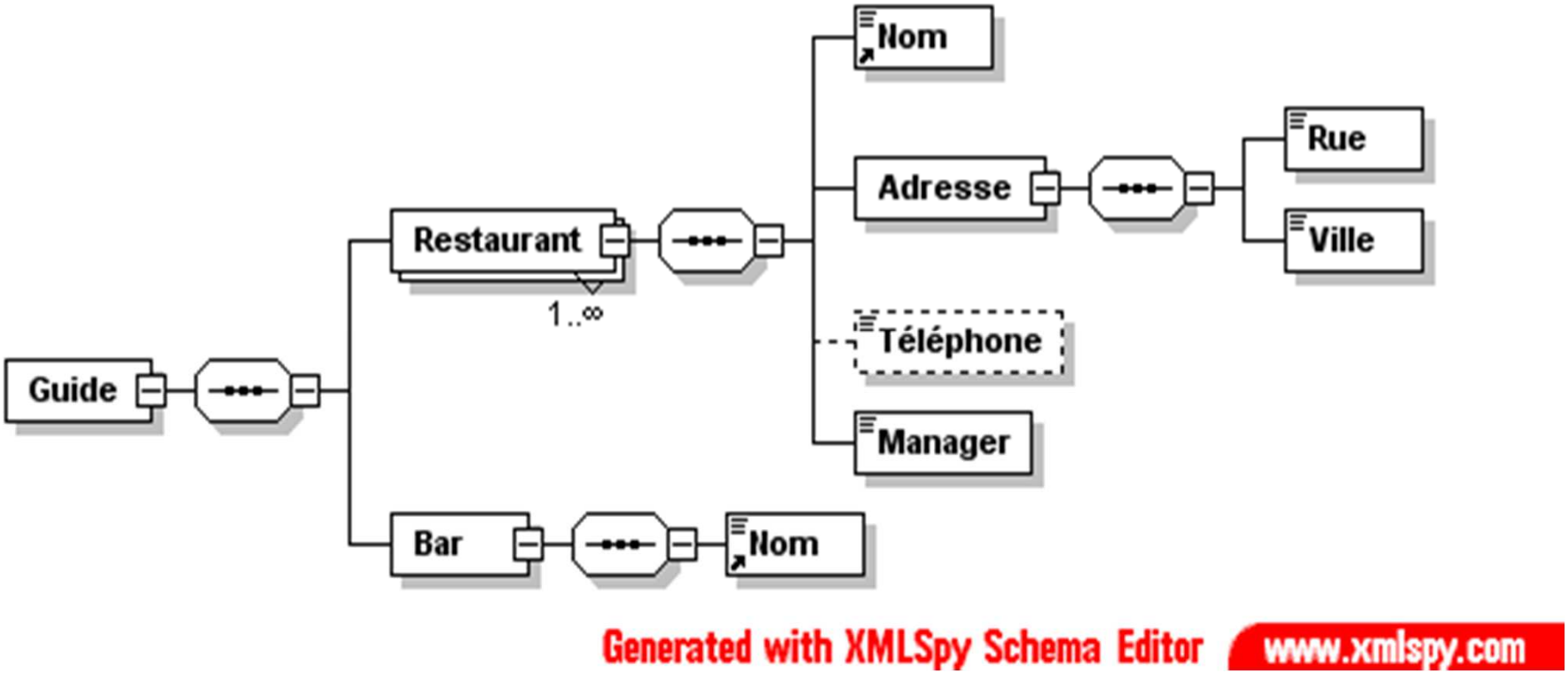
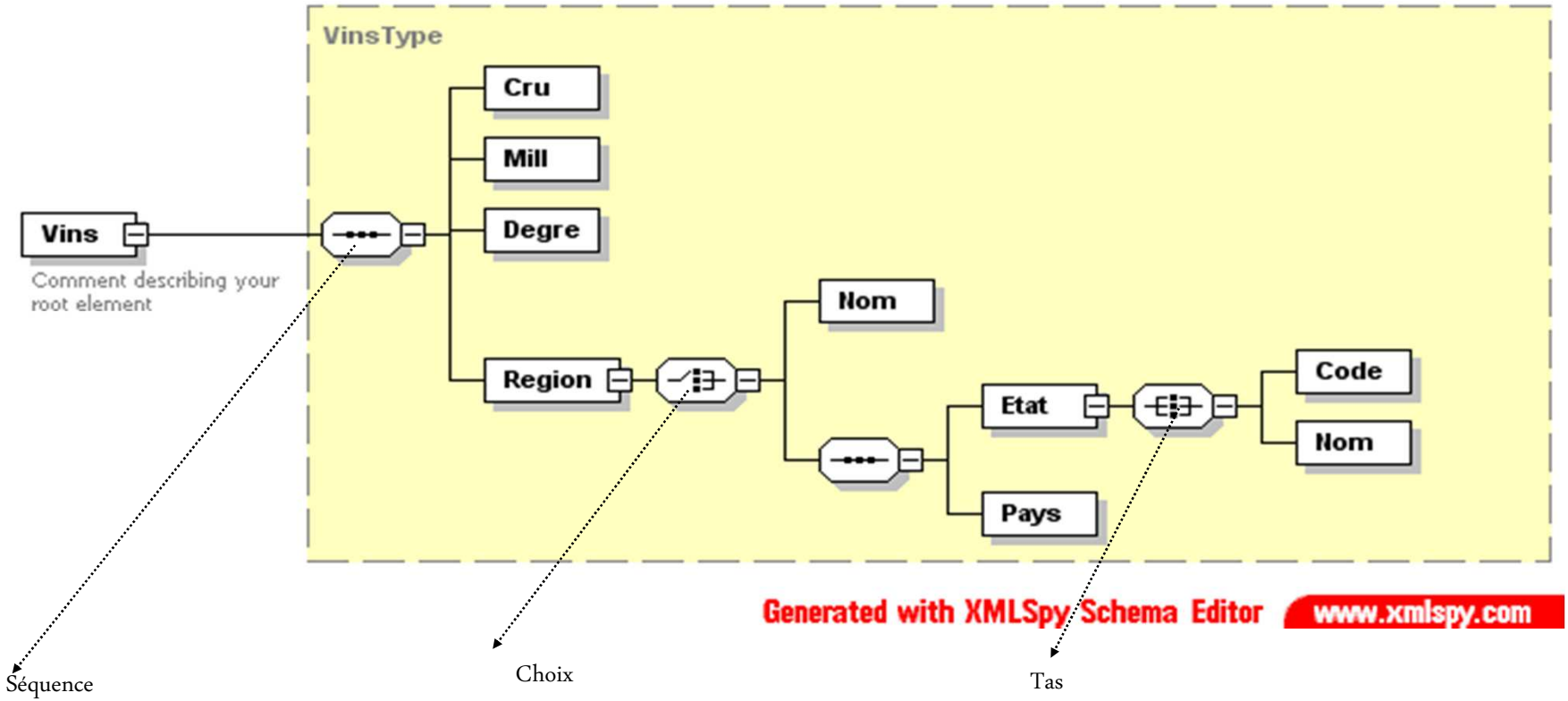


Diagramme de type (XML Spy)



Espace de nommage: Namesapce

- Les espaces de nommage (*namespace*) permettent de regrouper des éléments XML autour d'un nom unique.
- Des éléments portant un nom identique et définis dans des schémas différents peuvent cohabiter au sein d'un même document.
- Les éléments appartenant à un espace de nommage se distinguent des autres éléments par l'ajout d'un préfixe contenant l'URI de cet espace
- L'idée : utiliser des noms logiques associés aux namespaces au lieu de leurs URIs

Exemple : *<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0>*

Exemple d'utilisation :

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
        <xsl:template match="/">
                <html:balise xmlns:html="http://www.w3.org/TR/REC-html40">
                        </xsl:apply-templates/>
                </html:balise>
        </xsl:template>
</xsl:stylesheet>
```

Quelques outils de conception

Editeur	**Outil**	**Support**
Tibco (Extensibility)	XML Authority 2.0	DTD Schéma
Altova	XML Spy 5.0	DTD Schéma
Dasan	Tagfree 2000 DTD Editor	DTD
Data Junction	XML Junction	Schéma
Insight Soft.	XMLMate 2.0	DTD Schéma
Microstar Soft.	Near & Far Designer	DTD

Exercice

- Proposez un schema XML définissant un carnet de contacts
- Définir un document XML de contacts conforme à ce schéma
- Quels sont les avantages et les inconvénients des schémas XML ?

Bilan Schéma

- De plus en plus utilisées
  - Echange de modèles: XMI 2.0
  - Les services Web: SOAP, WSDL
  - …
- Le standard est un peu complexe

L’apport d’XML : Bilan (Partie I, II)

- Méta-langage ! Nombre non fini de :
  - **balises** et,
  - **attributs** associées
- Structuration arborescente !
- Représentation neutre, indépendamment des applications !

  (pas de sémantique d’applications sur les données)
- Séparation :
  - **données** (.xml)
  - **syntaxe logique** (.dtd ou .xsd)
- Formalisation :
  - **documents bien formées** *(syntaxe conforme à XML)*
  - **valides** *(structure conforme à une grammaire)*
- Outils et standards de manipulation pour les documents

XML

(Partie III)

Mise en forme, Traitement et Transformations des documents XML

Plan: Mise en forme

- Présentation des documents XML
  - Feuilles de styles CSS
  - Feuilles de styles XSL

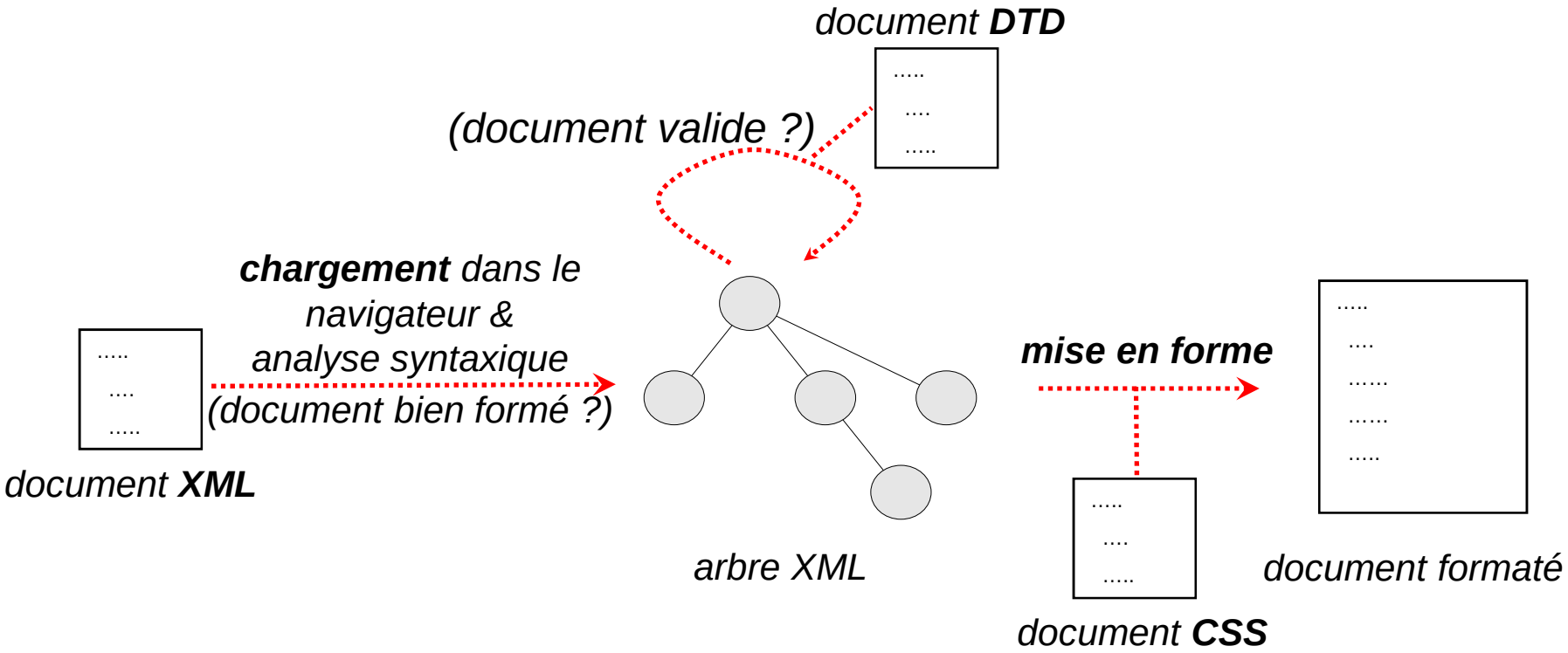
CSS : Rappel (Cascading Style Sheets)

- Présentation personnalisée des balises HTML (XML)
- Présentation par rapport à leur :
  - **nom** *(absolu ou relatif)*
  - **classe** (CLASS)
  - **identificateur** (ID)
  - **attributs** (en CSS2)
- Traitement en cascade !

  (toutes les balises descendantes héritent des propriétés de mise en forme de leurs balises mères)

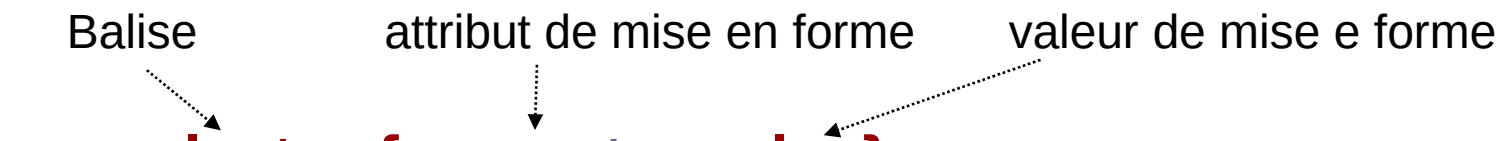
Présentation des documents XML avec CSS

- **CSS** *(Cascading Style Sheet)* ➔ associer une mise en forme à un document XML?



Syntaxe des règles CSS

- Forme générale:



selector {property: value}

Ou

appliquer la même mise en forme à plusieurs Balise

```
selector [, selector]* { property1: value1;
                         property2: value2;
                         ....
                         propertyN: valueN; }
```

- Exemple

```
H1 {text-align : center; color : blue}
```

Définition des styles CSS

- tous les Titres de niveau 1 et 2 <**H1**> et <**H2**>

```
H1, H2 { color: blue; text-decoration:underline; }
```

- toutes les <**B**> dans un <**P**>

```
P B {background-color: #CCCCCC; font-weight: bold }
```

- toutes les <**P**> de classe "***plain***" et toutes les balises de classe "**c1**"

```
P.plain {font-size:12 pt; line-height: 14pt;}
.c1 {font-size:12 pt; line-height: 14pt;}
```

- la balise ayant l'attribut **ID**="*fancy*"

```
#fancy {font-family:Arial; font-style: italic;}
```

- la balise <**book**> ayant **categorie**="*SF*" *(seulement dans CSS2)*

```
book[categorie="SF"] {display:none;}
```

Référencer une CSS

- Référencer une CSS dans un fichier XML:

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

- Référencer une CSS dans un fichier HTML:

  <link rel="stylesheet" type="text/css" href="path_to _file.css" />

- **Exemple**

MyCssFile.css

```
body { background-color: gray }
p { color: blue; }
h3 { color: white; }
```

MyHtmlFile.html

```
<html>
 <head>
  <link rel="stylesheet" type="text/css" href=" MyCssFile.css " />
 </head>
 <body> <h3> A White Header </h3>
<p> This paragraph has a blue font. The background color of this page is gray because we changed it with CSS!
    </p>
</body>
</html>
```

Exemple (1) XML & CSS

XML : exemple1.xml

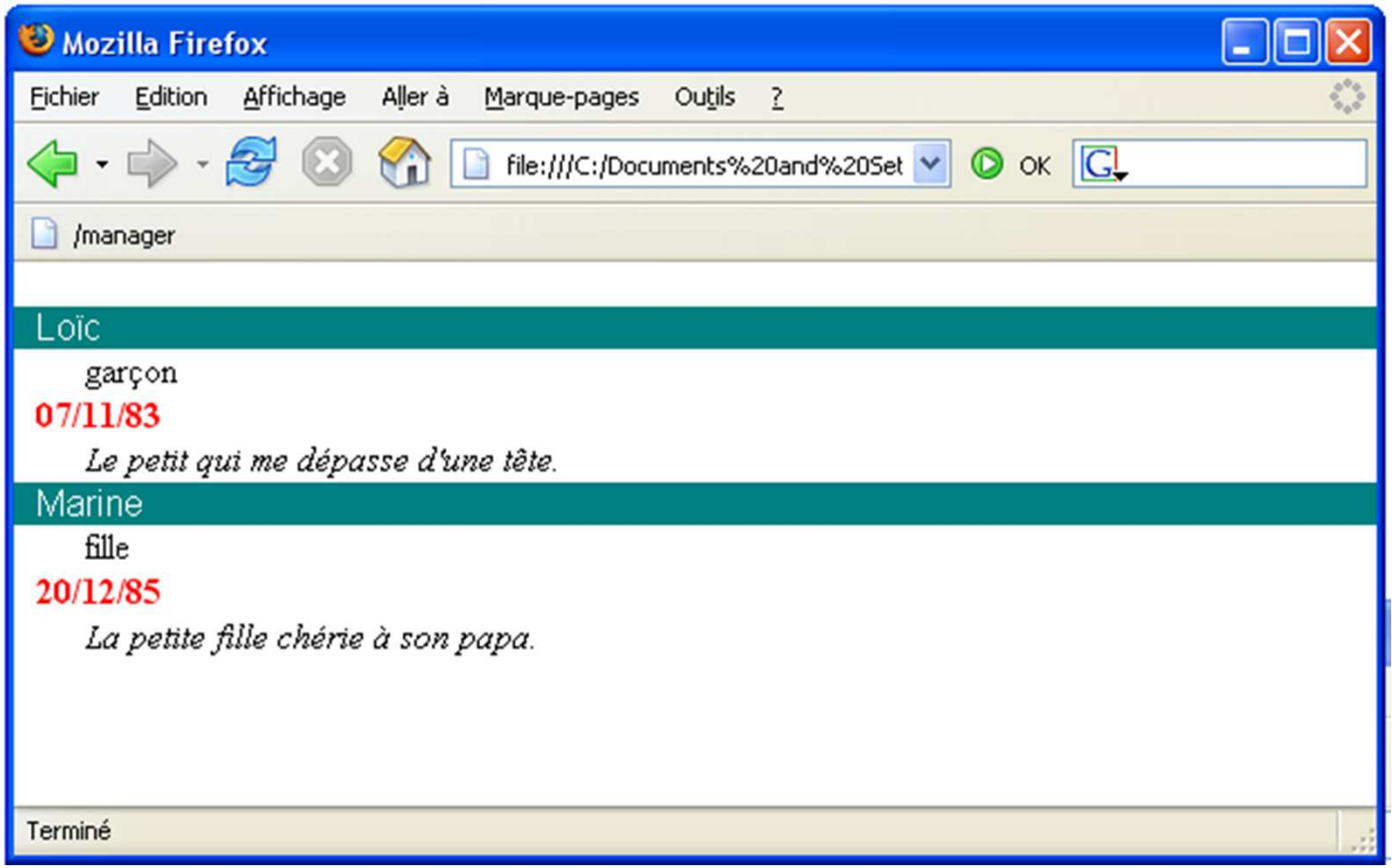
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="./exemple1.css" type="text/css" ?>
<racine>
   <enfant>
      <nom>Loïc</nom>
      <lien>garçon</lien>
      <date>07/11/83</date>
      <data>Le petit qui me dépasse d'une tête.</data>
   </enfant>
   <enfant>
      <nom>Marine</nom>
      <lien>fille</lien>
      <date>20/12/85</date>
      <data>La petite fille chérie à son papa.</data>
   </enfant>
</racine>
```

Référence vers la feuille CSS

CSS : exemple1.css

```
racine , enfant { }
nom  { display: block;    width: 250px;  font-size: 16pt ;  font-family: arial ;  font-weight: bold;
        background- color: teal;   color: white; padding-left: 10px; }
lien   {   display: block;  font-size: 12pt;  padding-left: 10px; }
date  { display: block; font-size: 12pt; color: red ; font-weight: bold; padding-left: 10px; }
data  { display: block; font-size: 11pt ; font-style: italic;   font-family: arial ; padding-left: 10px; }
```

Présentation dans un navigateur



Exemple (2) XML & CSS

Affichage des livres Science-Fiction d'un document XML en utilisant une feuille de style CSS annexe.

Fichier : exemple.xml

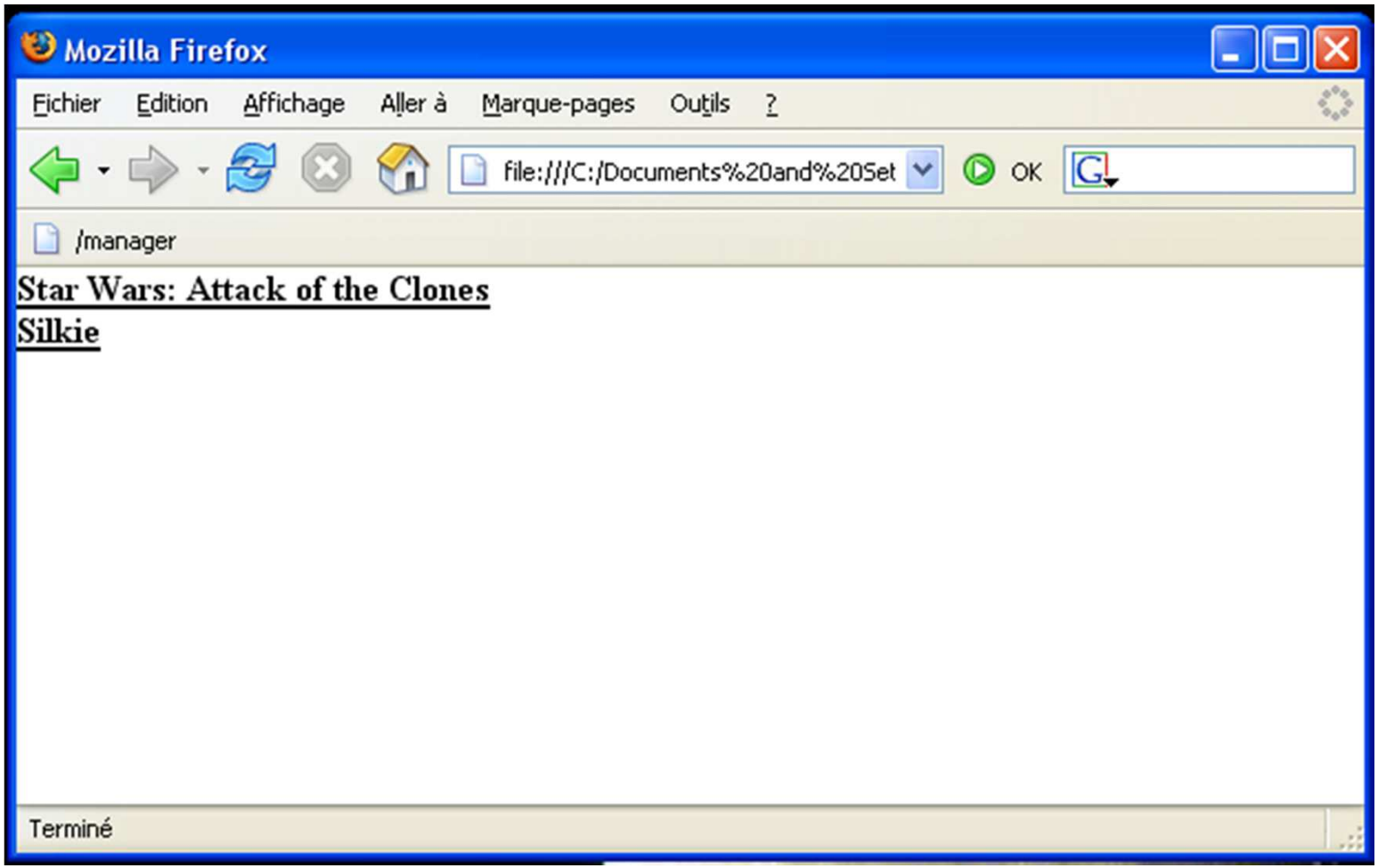
```
<?xml version="1.0"?>
<?xml-stylesheet href="./exemple2.css" type="text/css" ?>
<livres>
  <livre categorie ="SF">Star Wars: Attack of the Clones</livre >
  <livre categorie ="Literature">Le pendule de Foucault</livre >
  <livre categorie ="SF">Silkie</livre >
  <livre categorie ="Technique">Professional XML</livre >
</livres >
```

Référence vers la feuille CSS

Fichier : exemple.css

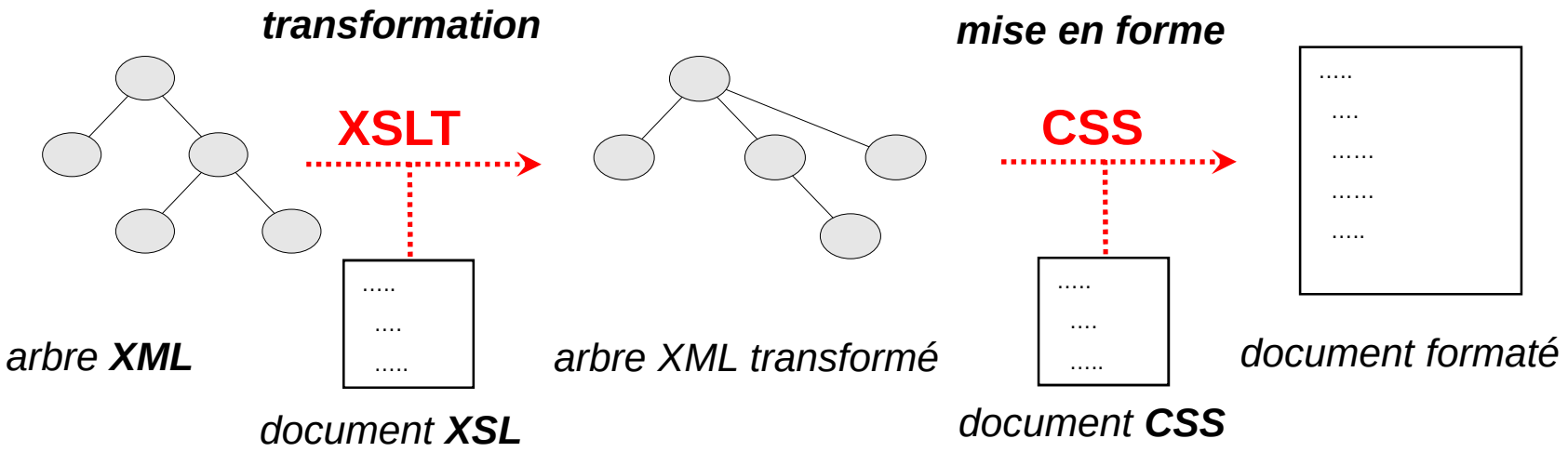
```
livre[categorie="Literature"] { display:none; }
livre[categorie="Technique"] { display:none;}
livre[categorie="SF"] { font-family:serif; font-size: 12pt;  line-height:14pt;
                        font-weight:bold; text-decoration:underline; display:block;}
```

Présentation dans un navigateur



XML StyleSheet Language (XSL)

- **XSL** = **CSS** + **XSLT** *(plus qu'une simple mise en forme …)*
- **CSS** → Mise en forme de fichier XML
- **XSLT** (XSLTransformation) → Transformation de l'arbre XML



Questions

- Proposer une CSS pour afficher un document XML de références bibliographiques.
- Quelle est l'utilité de la séparation entre les données et leurs mises en forme en XML ?
- Citez des exemples d'application utilisant les feuilles de styles CSS

Avantage de la séparation

contenu(.xml) / transformation (.xsl) / style (.css)

- Réutilisation des données
- Présentation personnalisée, réutilisation des formats
  - style standard ou personnel
  - selon les périphérique et les médias
  - etc.

L'API DOM

Plan: Traitement des documents XML

- Objectifs
- Structure de l'arbre XML
- Parcours des arbres XML
  - DOM *(Document Object Model)*
- Conclusions

Objectifs

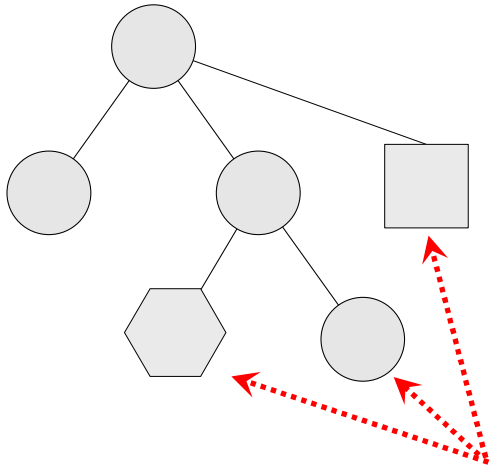
Accéder à l'information dans les documents XML **pour** :

- **récupérer**
  - les valeurs des **balises**,
  - les valeurs des **attributs**,
  - des **fragments d'un documents** XML)
- **par rapport à**
  - leur **type** *(instruction de traitement, commentaire, balise données ou attribut)*
  - leur **nom** *(dans le cas des balises et attributs)*
  - leur **position** *(avant, après, à coté une autre balise…)*
- **dans le but de**
  - **afficher leur contenu** *(e.g. dans un navigateur WEB)*
  - **exporter l'information** vers d'autres formats
- **selon une approche**
  - **générique et standard** → **réutilisable**
    (valable pour tous les langages de programmation !)
  - **simple** *(syntaxe lisible et efficace)*

Construction de l'arbre XML

document ***XML***

chargement
dans le navigateur
(document bien formé ?)



arbre XML = hiérarchie de <u>nœuds</u> de types différents

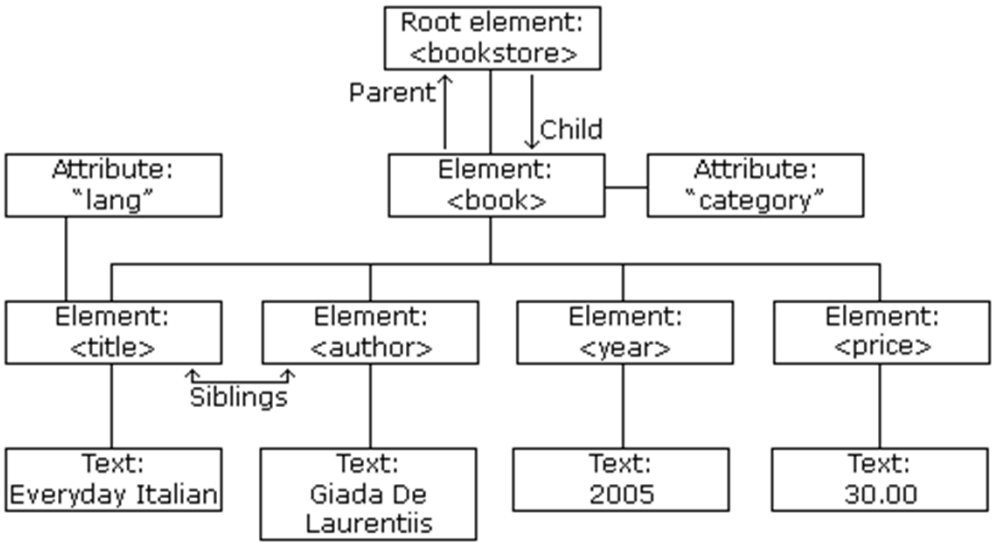
- **une racine** (le document lui-même),
- **instruction de traitement** (<? ….?>)
- **commentaire**, (<! --….-->)
- **élément**, (balises)
- **texte**, (le texte PCDATA d'une balise)
- **attribut**, (attribut dans une balise)

….. etc.

Exemple de la hiérarchie des nœuds DOM

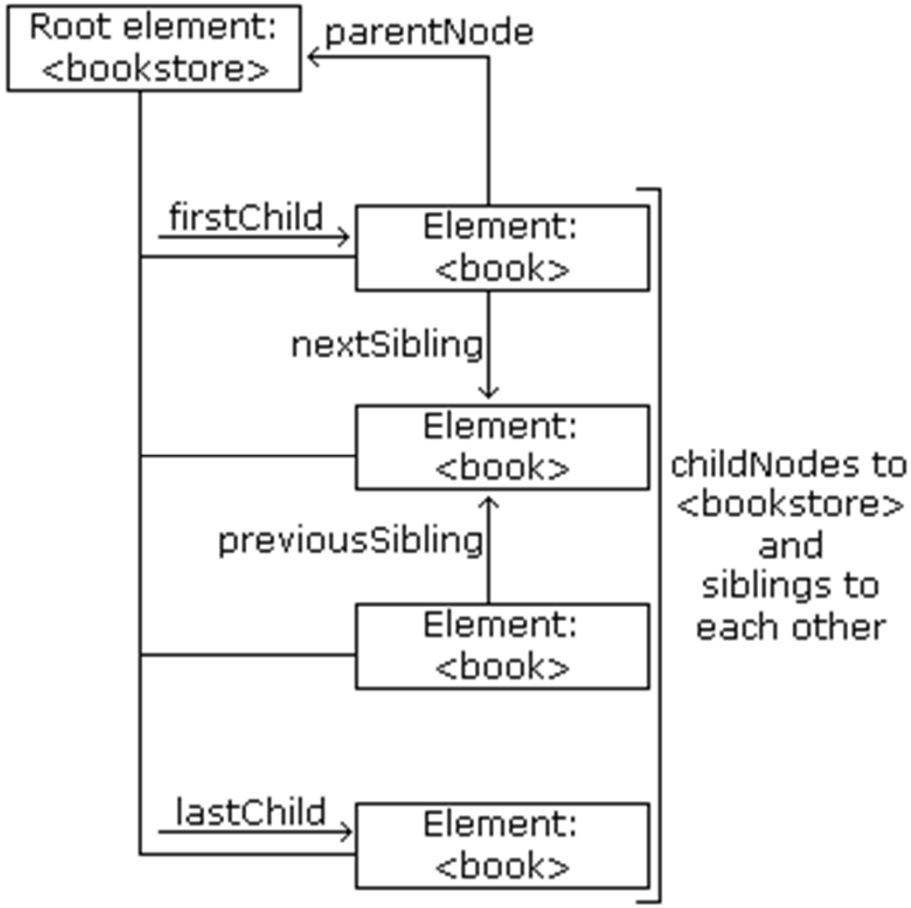
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
 <book category="COOKING">
   <title lang="en">Everyday Italian</title>
   <author>Giada De Laurentiis</author>
   <year>2005</year>
   <price>30.00</price>
</book>
<book category="CHILDREN">
   <title lang="en">Harry Potter</title>
   <author>J K. Rowling</author>
   <year>2005</year>
   <price>29.99</price>
 </book>
  …
<bookstore>
```

The corresponding Tree



Parcours d'un arbre de noeuds XML à l'aide de DOM

- **On peut naviguer entre les noeuds en utilisant les différentes relations qui peuvent exister entre eux:**
  - parentNode
  - childNodes
  - firstChild
  - lastChild
  - nextSibling
  - previousSibling



Structure de l'arbre XML en Noeuds

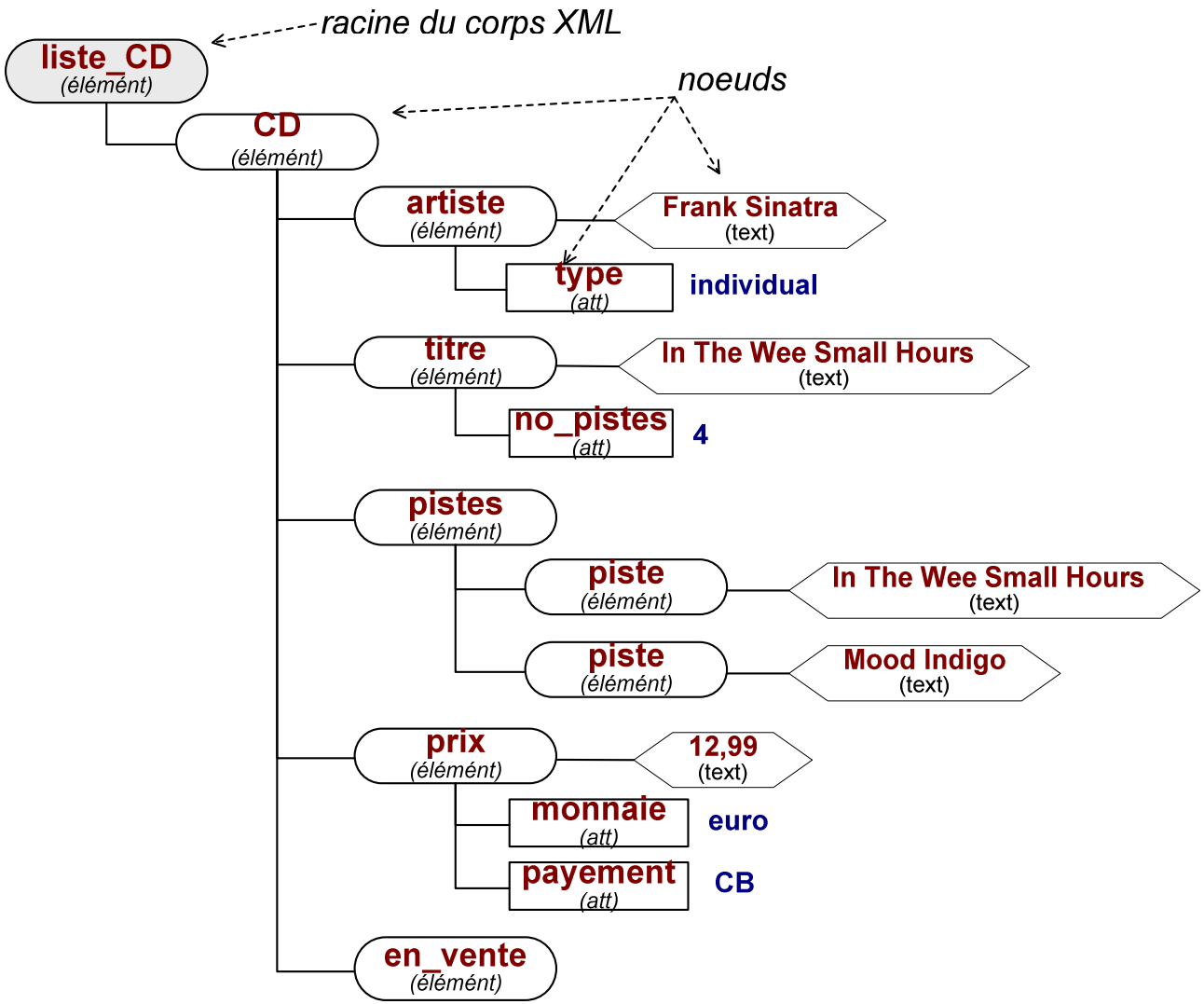
- **Un seul nœud "Document" contenant des nœuds fils :**
  - **"commentaire"** → *zéro ou plusieurs*
  - **"instructions de traitement"** → *zéro ou plusieurs*
  - **"DOCTYPE"** → *au plus un, la déclaration DOCTYPE dans le prologue XML*
  - **"racine"** → *un, le corps du document XML*

- **Les nœuds fils d'une "racine" sont de type :**
  - **"element"** → balises filles
  - **"texte"** → le texte entre les deux balises

    NB: Il peut y avoir plusieurs nœuds texte si le contenu est mélangé avec des balises
  - **"commentaire"** → *zéro ou plusieurs*
  - **"instructions de traitement"** → *zéro ou plusieurs*
  - **"section CDATA"** → *texte non interprété entre* [….]

- **L'ordre des nœuds voisins est celui de la lecture du document !**

Exemple d'arbre XML

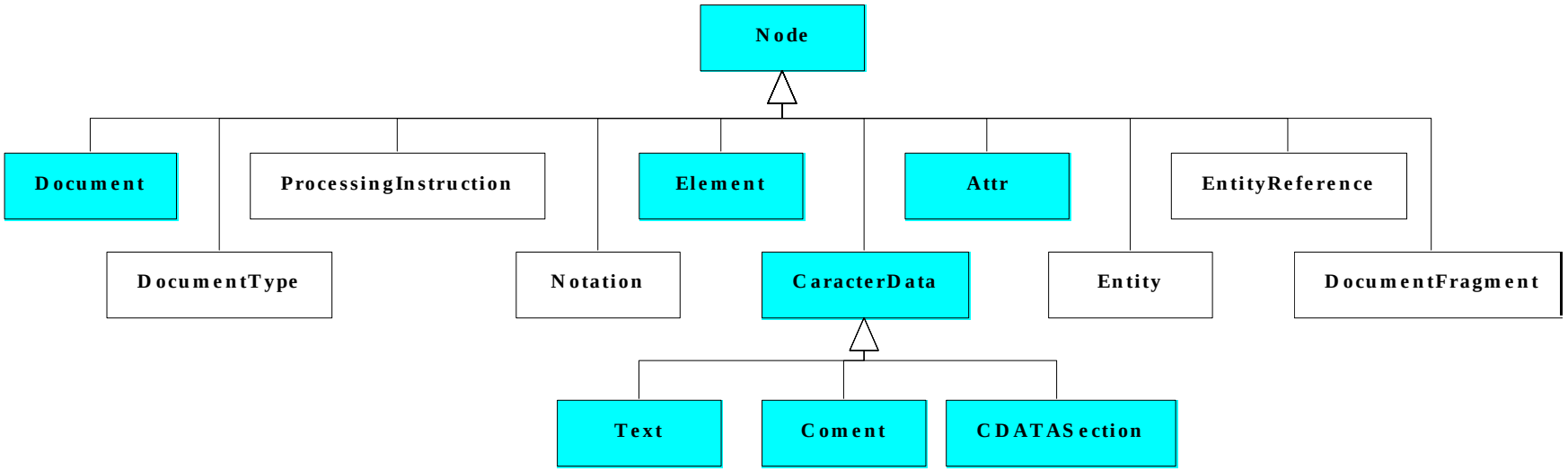


Parcours des arbres XML en utilisant DOM *(Document Object Model)*

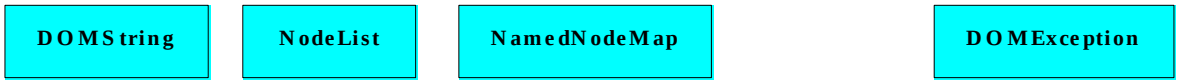
- Une **API standard** pour le parcours d'arbres XML !
- **Indépendante du langage de programmation** utilisé !
- Définie une **hiérarchie de classes** pour le traitement des nœuds XML !
- Le "**Nœud**" est la classe DOM principale *(voir diapo. suivant)*
- Chaque **objet DOM** définit :
  - **Des propriétés** pour les nœuds *(accès en lecture seule ou lecture-écriture),*
  - **Des méthodes** de traitement.
- Son utilisation repose sur l'existence d'un **parseur XML** !

Les classes DOM (1)

- La Hiérarchie de classes DOM de niveau 1



- Types de données auxiliaires



Légende :

Interfaces étudiées dans ce cours

Les classes DOM (2)

Classes DOM de base :

- **Node** → **classe de base** *(tout élément dans l'arbre XML est un nœud)*
- **Document** → **le document XML entier** avec le **Prologue** et le **Corps**
- **CDATASection** → **section CDATA** *(texte non interprété) (<![CDATA[...]>)*
- **Comment** → **commentaire** correspondant au *(e.g. <!--....-->)*
- **Element** → une **balise** *(e.g. <balise>)*
- **Attr** → un **attribut** dans une balise
- **Text** → **contenu textuel** d'une balise *(e.g.<balise>….. </balise>)*

Classes DOM auxiliaires :

- **NodeList** → une **liste ordonnée** de nœuds
- **NamedNodeMap** → **ensemble non ordonné** de nœuds *(accès par nom)*
- **DOMException** → **exception** de traitement de l'arbre XML

Autres nœuds :

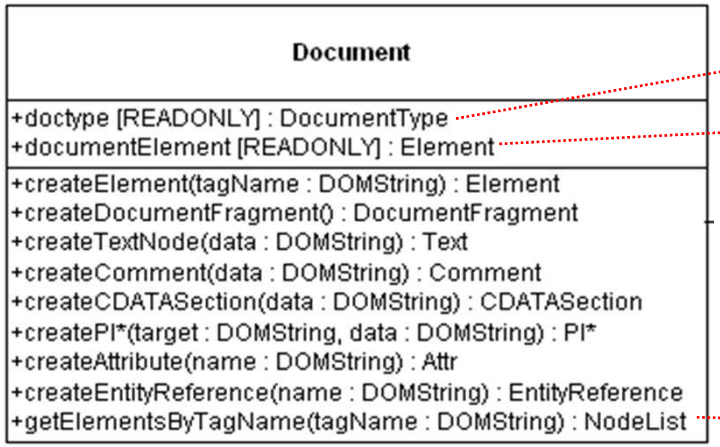
- DocumentType, ProcessingInstruction, Notation, Entity, EntityReference, DocumentFragment .....

La classe ***Document***

- Représente le document XML

attributs
(READONLY = en lecture seule)

méthodes



Type du document

La racine du document

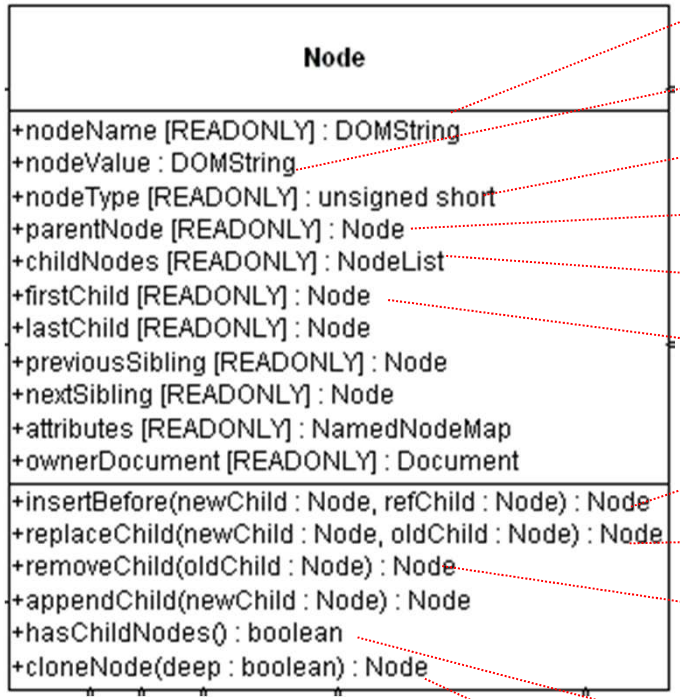
Rechercher une balises par nom

- Syntaxe des **:**
  - **attributs :** *nom_de_l'attribut [accès] : type_de_l'attribut*
  - **méthodes :** *nom_de_la_méthode_publique (paramètres) : type_de_la_valeur_de_retour*

nom_parametre : type_parametre

La classe ***Node*** (1)

- Toute branche ou feuille dans l'arbre XML est un nœud.
- On peut donc parcourir l'arbre XML de nœud en nœud !



*Le **nom du nœud** (i.e. nom de la balise, de l'attribut etc.)*

*La **valeur du nœud** (i.e. contenu de la balise, l'attribut etc.)*

*Le **type du nœud** (**1**=Element; **2**=Attribut; **3**=Text; **4**=CDATASection; **9**=Document)*

*Le **nœud parent***

*La **liste des nœuds fils***

*Le **premier nœud fils dans la liste** ……etc.*

Insérer un nœud avant

Remplacer un nœud fils

Effacer un nœud fils

a-t-il des nœuds fils ?

Dupliquer par valeur le nœud (cloner)

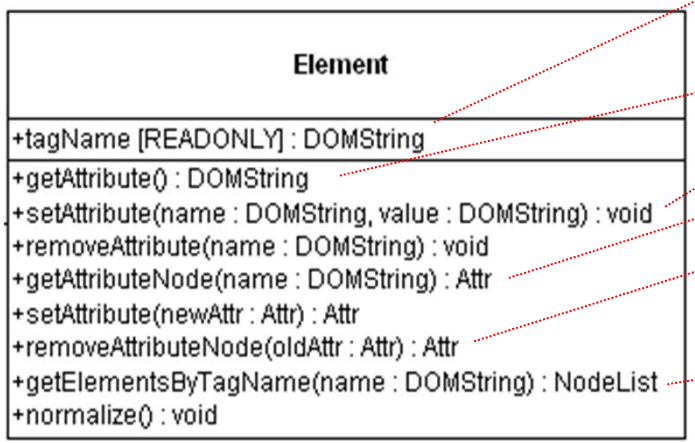
La classe ***Node*** (2)

La valeurs des **nodeName**, **nodeValue** et **attributs** dépend de chaque type de nœud

Type de Noeud	nodeName	nodeValue	attributes
Attr	name of attribute	value of attribute	null
CDATASection	"#cdata-section"	content of the CDATA Section	null
Comment	"#comment"	content of the comment	null
Document	"#document"	null	null
DocumentFragment	"#document-fragment"	null	null
DocumentType	document type name	null	null
Element	tag name	Null ?	NamedNodeMap
Entity	entity name	null	null
EntityReference	name of entity referenced	null	null
Notation	notation name	null	null
ProcessingInstruction	target	entire content excluding the target	null
Text	"#text"	content of the text node	null

La classe ***Element***

- Représente une **balise** XML



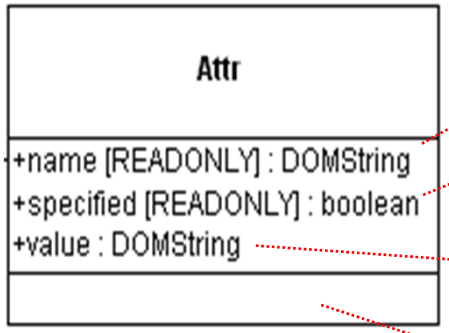
*Le **nom de la balise (tag)** (e.g. HTML)*

Extraire la valeur d'un attribut par son nom
Affecter une valeur à un attribut par son nom
*Récupérer **l'attribut** comme Node*
Effacer un attribut

*Chercher l'**Element** avec le "**nom**" dans la **liste des nœuds fils***

La classe ***Attr*** (attribut)

- Représente un attribut d'une balise XML



*Le **nom de l'attribut** (e.g. "style")*

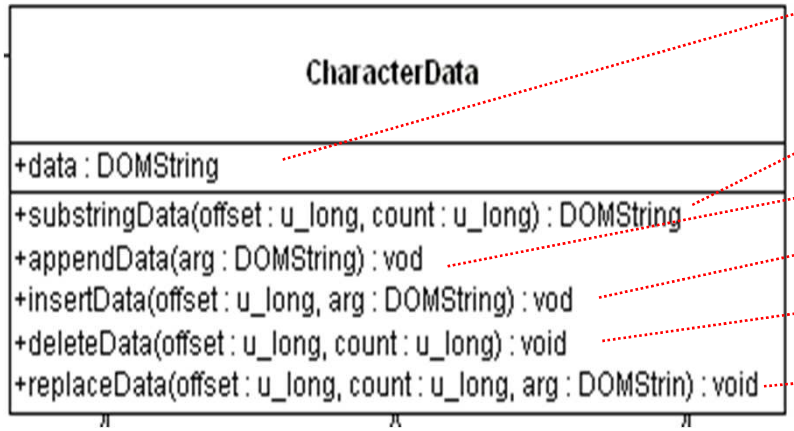
Si la valeur de l'attribut est celle du document d'origine ou a été modifiée

*La **valeur de l'attribut** (e.g. "133" ou "" si vide)*

….. il n'y a pas des méthodes
(méthodes héritées de la classe Node)

La classe ***CharacterData***

- Une **classe abstraite** qui représente des **opérations sur les nœuds de type chaîne de caractères** !



Le contenu textuel

Soustraire une sous chaîne de caractères

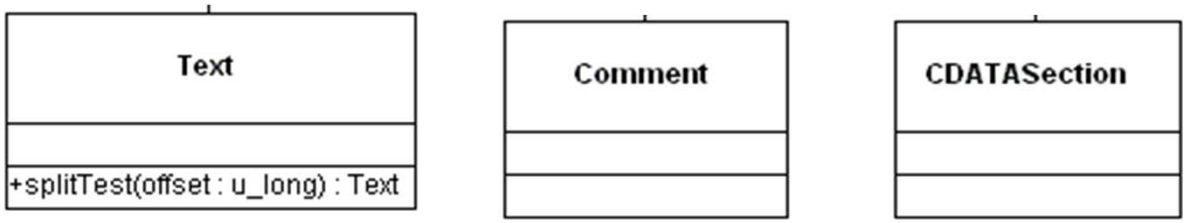
Ajouter la chaîne donnée à la fin

Insérer la chaîne donnée à la position offset

Effacer le contenu

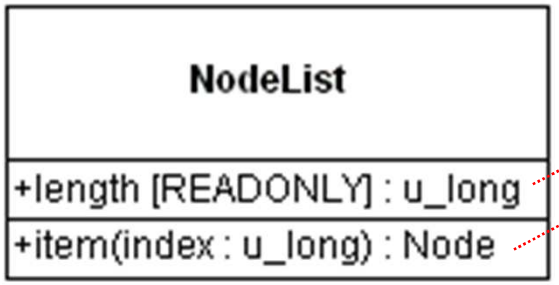
Remplacer par une autre chaîne une partie du contenu

- Classes dérivées : **Texte, Comment et CDATASection**



Classes DOM auxiliaires

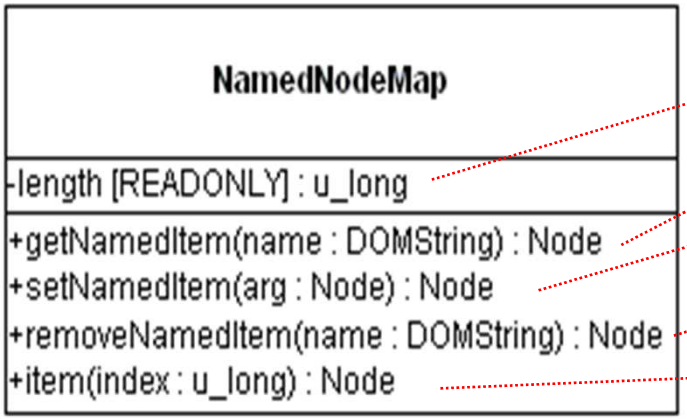
- Une liste ordonnée de nœuds : l'accès par index



Nombre d'éléments dans la liste de noeuds

Le nœud avec l'index …

- Un ensemble non ordonnée des nœuds : l'accès par nom



Nombre d'éléments dans la liste de noeuds

Récupérer un nœud *par son nom.*

Insérer *un Nœud*

Effacer *le nœud avec le "nom"*

Récupérer *un nœud avec l'index ….*

(dans l'ordre de construction de la Liste)

Précisions pour l'API DOM

- **L'accès aux attributs dépend du langage d'implémentation :**
  - Par **nom**
  - En utilisant les méthodes d'accès : ***getNom()*** et ***setNom ()*** où "***Nom***" représente le nom de l'attribut

- DOM **ne standardise pas une méthode pour créer l'arbre d'un document XML** (dépend d'un parseur à l'autre)

Etapes pour l'utilisation de DOM en Java: parcourir un document XML (1)

1. Importer les package XML:

```
import org.w3c.dom.*;
import javax.xml.parsers.*;
import java.io.*;
```

2. Créer un Constructeur de parseurs …

```
DocumentBuilderFactory factory =DocumentBuilderFactory.newInstance();
```

…puis un parseur:

```
DocumentBuilder builder = factory.newDocumentBuilder();
```

3. Créer un Document à partir d'un fichier ou d'un flot:

```
Document document= builder.parse(newFile(file));
```

Etapes pour l'utilisation de DOM en Java: parcourir un document XML (2)

4.Extraire l'élément racine

```
Element root = document.getDocumentElement();
```

Ainsi vous pouvez :

5.Examiner les attributs

getAttribute("attributeName") retourne un attribut spécifique

getAttributes() retourne une Map (table) de noms/ valeurs des attributs

6.Examiner les sous-elements

getElementsByTagName("sub-elementName") retourne la liste des sous-éléments spécifiques

getChildNodes() retourne la liste de tous les nœuds fils et petits fils et …

Les deux méthodes retournent des liste de Node et non pas de Element

» Comme Node est le parent de Element …

- … les résultats de getElementsByTagName peuvent être directement castés en Element
- … les résultats de getChildNodes sont des noeuds de différents types et ne peuvent donc pas être directement castés en Element

Exemple d'utilisation de DOM en Java (1)

- **Exemple de parcours de fichier XML**

Le fichier XML en input :

```
<?xml version="1.0" encoding="UTF-8"?>
<dataroot>
<ref_biblio>
   <ref>Globe99</ref>
   <type>article</type>
   <author>Van Steen, M. and Homburg, P.  and Tanenbaum, A.S.</author>
   <title>Globe: A Wide-Area Distributed System</title>
 </ref_biblio>
 <ref_biblio>
   <ref>ada-rm</ref>
   <type>techReport</type>
   <author>International Organization for Standardization</author>
   <title>Ada Reference Manual</title>
 </ref_biblio>
</dataroot>
```

Comment accéder à ces infos ?

Exemple d'utilisation de DOM en Java (2)

```
public class Exemple {
  public static void main(String[ ] args) {

    /*   instancier le constructeur de parseurs   */
    DocumentBuilderFactory _factory = DocumentBuilderFactory.newInstance();

    /*   ignorer les commentaires dans les fichiers XML parsés   */
    _factory.setIgnoringComments(true);

    /*   créer un parseur à partir de l'instance du constructeur de parseurs   */
    DocumentBuilder _builder = _factory.newDocumentBuilder();

    /*   charger le document en utilisant le parseur   */
    String filepath = ".\\bib.xml";
    Document doc = _builder.parse(filepath);

    /*    Parser le document    */
    Element library = doc.getDocumentElement();
```

élément racine

Partie fixe pour chaque utilisation du DOM en Java

Exemple d'utilisation de DOM en Java (3)

```
/*  Récupération des infos à partir de l'élément document */

/* récupérer la liste de tous les nœuds (fils et petits fils et …)    */
NodeList allChilds = library.getChildNodes();

/* parcourir la liste  */
for (int i = 0; i <  allChilds.getLength() ; i++) {
    Node node = allChilds.item(i);

      /* vérifier si le Node est un Element (Balise) et pas un Att ou un Text    */
      if (node.getNodeType() == Node.ELEMENT_NODE) {

          / * caster le Node en un Element   */
          Element elt = (Element) node;

          /*  récupérer le Element (Balise) title   */
          Element title = (Element) elt.getElementsByTagName("title").item(0) ;

          /*  récupérer son seul Node fils qui est de type Text   */
          Node text = title.getFirstChild() ;

          /* récupérer la valeur du Node text qui représente l'info que l'on cherche */
          String titre = text. getNodeValue();
          }
}}}
```

Exercice

- Soit le document XML suivant

```
<AAA>
  <BBB/>
  <BCD/>
  <CCC>
    <BBB nom='titi' />
    <BBB nom='toto' />
  </CCC>
</AAA>
```

- Ecrire le code DOM pour la récupération de la valeur de l'attribut *nom* de la balise <BBB>

- Quelles sont vos critiques sur l'API DOM

Conclusion DOM

- API standardisée (W3C)
- Construction d'un arbre d'objets XML :
  - Facile à programmer
  - API indépendante du langage de programmation
  - Lourd (code long) SkyTeam
  - Parcours de l'arbre peut s'avérer ardu
- Nous souhaitons donc simplifier le parcours des arbres XML
  - XPath

XML and DOM Resources

- Java API Docs
  http://java.sun.com/j2se/1.5.0/docs/api/
  http://java.sun.com/j2se/1.5.0/docs/api/org/w3c/dom/Node.html
  http://java.sun.com/j2se/1.5.0/docs/api/org/w3c/dom/Element.html
- XML 1.0 Specification
  http://www.w3.org/TR/REC-xml

- WWW consortium's Home Page on XML
  http://www.w3.org/XML/

Sun Page on XML and Java
  http://java.sun.com/xml/

O'Reilly XML Resource Center
  http://www.xml.com/

L'API SAX

SAX (Simple API for XML)

- Un *org.xml.sax.XMLReader* parse les documents XML et génère des événements (rencontre d'un nouveau tag, rencontre d'une erreur, etc..)

Fonctionnement par événements et «call-backs»:

1. Enregistrement de handlers pour la personnalisation du traitement des évènements du parseur en y attachant des instances des interfaces *org.xml.sax.ContentHandler*, *org.sax.xml.ErrorHandler*, etc..
2. Ainsi le parseur invoque les méthodes du Handler lorsqu'un évènement arrive lors de la lecture du document XML:
   - startElement()
   - endElement()
   - characters()
   - startDocument()
   - endDocument()

Exemple de parseur SAX

```
public class MySAXParser {
 public static void main(String[] args) {
      try {
           Class c = Class.forName("org.apache.xerces.parsers.SAXParser");
            XMLReader reader = (XMLReader)c.newInstance();
            MySAXHandler handler = new MySAXHandler();
           reader.setContentHandler(handler);
           reader.parse("test.xml"); /** Lecture d'un document
      } catch(Exception e){e.printStackTrace();}
   }
 }
class MySAXHandler extends DefaultHandler {
     private String tagCourant = ""; /* Actions a réaliser lors de la détection d'un nouvel
     élément. */
     public void startElement(String nameSpace, String tagName, String qName,
     Attributes attr) throws SAXException {
           tagCourant = tagName;
            System.out.println("Debut tag : " + localName);
     }
}
```

Implémentation apache de parseurs SAX

Attribution du gestionnaire de contenu définit par le développeur

Analyse de SAX

Points positifs:

- Convient pour validation de formatage d'un document XML
- Convient pour les applications voulant récupérer les informations depuis quelques champs d'un document XML en entrée sans se soucier de la validité du document. Exemple: une servlet cherchant à récupérer un paramètre depuis un document XML
- Très rapide
- Supporte le traitement des fichiers DTD
- Une faible consommation en mémoire (le document XML n'est pas entièrement chargée en mémoire)

Points négatifs:

- Impossibilité de reculer dans la lecture d'un document
- Manque de vision synthétique du document: ne convient pas pour les applications graphiques
- Mise en oeuvre très lourde si besoin de valider le document XML

➔ API DOM pour une plus grande flexibilité dans la manipulation et la modification du document traité

Xpath

Parcours d'arbres XML avec XPath

- **XPath** = Expressions des chemins de parcours dans l'arbre XML
- Objectifs :
  - **seulement** pour **l'extraction d'informations** (+ par rapport à DOM)
  - **syntaxe simple et courte** (+ par rapport à DOM)

- Comparaison : XPath est pour XML ce qui est l'Expression de chemins de fichiers pour un système de fichiers !

Expressions XPath : Notions de base

- Deux types de chemins de localisation : **- relatifs** **- absolus**

- **Chemin relatif :**
  - le nœud de départ est le **nœud courant**
  - une **séquence d'une ou plusieurs étapes** séparées par le caractère « ***/*** ».
  - **chaque étape sélectionne un ensemble de noeuds** relativement au noeud contextuel

    Format : étape1/étape2/….étapeN

- **Chemin absolu :**
  - le nœud de départ est le nœud racine
  - **commence par** « **/** » **qui sélectionne la racine de l'arbre** …
  - …suivi par un chemin relatif

    Format : /étape1/étape2/….étapeN

Etapes de localisation

Une étape de localisation se fait en **3 temps** :

- **un axe** → le sens de la recherche
- **un noeud de test** → le **type du noeud** et le **nom** des noeuds obtenus par l'étape de localisation
- **0 ou n prédicats** → expressions arbitraires pour raffiner (filtrer) le résultat.

Exemple : **child::para[position()=1]**

noeud de test

l'axe

un prédicat

Les **axes** de recherche

- **self** : le nœud courant lui-même ;
- **child** : les enfants du nœud courant ;
- **descendant, descendant-or-self** : tous les descendants du nœud courant, + nœud courant ;
- **parent** : le père du noeud courant ;
- **ancestor, ancestor-or-self** : les ancêtres du nœud courant, + nœud courant ;
- **attribute** : les attributs du nœud courant ;
- **preceding, following** : les nœuds, précédents ou suivants, du noeud courant, dans l'ordre de lecture du document (*pas de descendants ou d'ancêtres*);
- **preceding-sibling, following-sibling** : les frères, précédant ou suivant, le nœud courant ;
- **namespace** : les espaces de noms.

Les **noeuds** de test

- **Nom** → le nom du nœud cherché
- ***** → "Joker" = tous les nœuds
- **Type** →
  - **node()** → tous les nœuds de tout type
  - **text()** → tous les nœuds de type text
  - **comment()** → tous les nœuds de type commentaire
  - **processing-instruction()** → tous les nœuds de type instruction de traitement
  - **processing-instruction(name)** → le nœud nommé de type instruction de traitement

Les **prédicats**

- Syntaxe : ‘[‘ Expression ‘]’
- Quelques expressions
  - position()=3
  - position()=last()-1
  - attribute::type=‘warning’
- Prédicats peuvent être chaînés [ ][ ]
- Quelques fonctions intéressantes :

count(*node-set*)	nombre des nœuds de la liste
id(*object*)	le nœud ayant l’ID donné
position(nombre)	nœud avec la position spécifié dans la liste
last()	le dernier nœud dans la liste sélectionné

Exemples d'expressions XPath
Syntaxe non-abrégée (1)

- **child::para** sélectionne l'élément **para** enfant du noeud contextuel
- **child::*** sélectionne tous les éléments enfant du noeud contextuel
- **child::text()** sélectionne tous les noeuds textuels du noeud contextuel
- **attribute::name** sélectionne l'attribut **name** du noeud contextuel
- **attribute::*** sélectionne tous les attributs du noeud contextuel
- **descendant::para** sélectionne tous les descendants **para** du noeud contextuel
- **ancestor-or-self::div** sélectionne tous les ancêtres **div** du noeud contextuel et le noeud contextuel lui-même si c'est un **div**
- **child::*/child::para** sélectionne tous les petits enfants **para** du noeud contextuel
- **/** sélectionne la racine du document (qui est toujours le parent de l'élément document)

Exemples d'expressions XPath
Syntaxe non-abrégée (2)

- **/descendant::olist/child::item** sélectionne tous les éléments **item** qui ont un parent **olist** et qui sont dans le même document que le noeud contextuel
- **child::para[position()=last()-1]** sélectionne l'avant dernier **para** enfant du noeud contextuel
- **following-sibling::chapter[position()=1]** sélectionne le prochain **chapter** cible du noeud contextuel
- **/child::doc/child::chapter[position()=5]/child::section[position()=2]** sélectionne la 2ième **section** du 5ième élément **chapter** de l'élément **doc**
- **child::para[attribute::type='warning'][position()=5]** sélectionne le 5ième enfant **para** du noeud contextuel qui a un attribut type dont la valeur est **'warning'**
- **child::chapter[child::title='Introduction']** sélectionne l'enfant **chapter** du noeud contextuel qui a un ou plus enfant **title** avec un contenu textuel égal à **'Introduction**'
- **child::*[self::chapter or self::appendix][position()=last()]** sélectionne le **dernier** enfant **chapter** ou **appendix** du noeud contextuel

Abréviations

- child peut être **implicite**
  - child::div/child::para ➔ div/para

- attribute peut être remplacé par @
- /descendant-or-self::node()/ peut être remplacé par //
- self.node() peut être remplacé par .
- parent::node() peut être remplacé par ..

Exemples d'expressions XPath
syntaxe abrégée (1)

- ***** sélectionne **tous les enfants** du noeud contextuel
- **text()** sélectionne tous les noeuds enfants de type textuels du noeud contextuel
- **@name** sélectionne **l'attribut name** du noeud contextuel
- **@*** sélectionne **tous les attributs** du noeud contextuel
- **para[1]** sélectionne le **premier** enfant para du noeud contextuel
- **para[last()]** sélectionne le **dernier** enfant para du noeud contextuel
- ***/para** sélectionne tous les petits enfants **para** du noeud contextuel
- **/doc/chapter[5]/section[2]** sélectionne la 2ième section du 5ième **chapter** de **doc**
- **chapter//para** sélectionne les descendants **para** des enfants **chapter** du noeud contextuel
- **//para** sélectionne tous les descendants **para** de la racine du document et donc, sélectionne tous les éléments **para** contenant le noeud contextuel

Exemples d'expressions XPath
syntaxe abrégée (2)

- **.** sélectionne le noeud contextuel
- **.//para** sélectionne les descendants **para** du noeud contextuel
- **.. sélectionne** le parent du noeud contextuel
- **para[@type="warning"][5]** sélectionne le 5ième enfant **para** du noeud contextuel qui a un attribut type ayant la valeur **'warning'**
- **para[5][@type="warning"]** sélectionne le 5ième enfant para du noeud contextuel si celui-là a un attribut type dont la valeur est warning
- **chapter[title="Introduction"]** sélectionne les enfants **chapter** du noeud contextuel qui ont au moins un enfant **title** dont le contenu textuel (string-value) est **'Introduction'**
- **employee[@secretary and @assistant]** sélectionne tous les enfants **employee** du noeud contextuel qui ont simultanément les attributs **secretary** et **assistant**

Exercice

- Soit le document XML suivant

```
<AAA>
  <BBB/>
  <BCD/>
  <CCC>
    <BBB nom='titi' />
    <BBB nom='toto' />
  </CCC>
</AAA>
```

- Ecrire le chemin XPath pour la récupération de toutes les balises <BBB>
- Ecrire le chemin XPath pour la récupération de la valeur de l'attribut nom de la balise <BBB>
- Quelles sont vos critiques de XPath

Bilan général sur DOM et XPaths

- Pour le parcours de l'arbre XML XPath est plus simple que DOM !
- DOM est plus complet que XPath car il permet la création des arbres XML !
- DOM & XPath sont des standards W3C !
- Indépendants du langage de programmation !
- La transformation des documents XML est possible également à l'aide d'autres standards (voir XSLT)
- …

XML

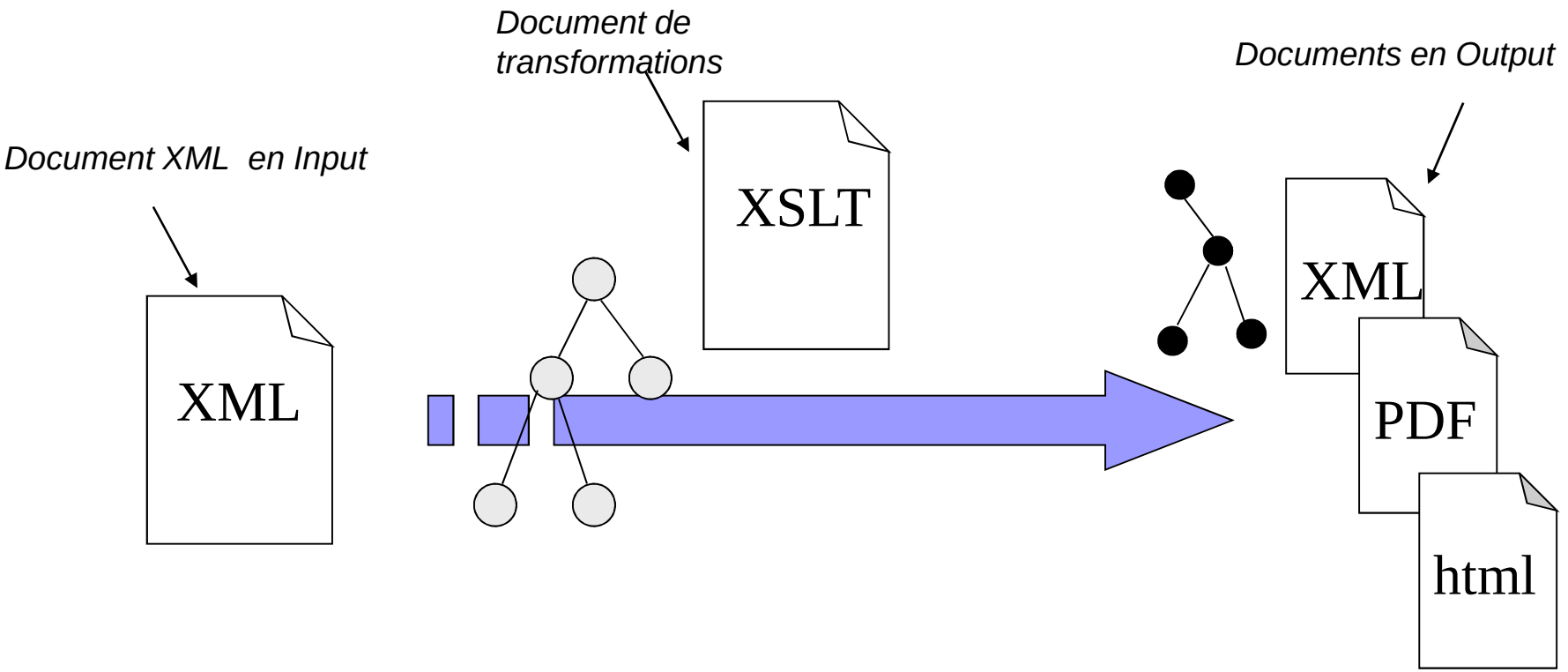
(Partie II)

Transformations de documents XML avec XSLT

XSLT e**X**tensible **S**tylesheet **L**anguage **T**ransformation ?

- XSL Transformation
- Langage de **transformation d'arbre XML**
- **Syntaxe conforme à XML**
- Fonctionnement par **pattern-matching**
- W3C Recommandation Novembre 1999

Principe de Fonctionnement



Templates XSLT: Principes

- **Feuille de style XSL = Ensemble de Règles de transformation (Templates)**
- Chaque **Template** est défini par le couple :
  - **Identification** en utilisant XPath d'un noeud dans XML source, e*x:* Balise XML
  - **Traitement** = écriture en sortie, ex*:* générer un nœud XML

Ex:

```
<xsl:template match="/">
        <html>
            <head></head>
                <body>
                    <xsl:apply-templates/>
                </body>
        </html>
</xsl:template>
```

Identification

Traitement : écrire ces éléments en sortie

Traitement XSLT: Principes

- **Processeur XSLT fonctionne en trois temps**
  1. Charger le document source en mémoire et construire une liste de nœuds de l'arbre XML
     - **Initialement** cette liste ne contient que **le nœud racine**
  2. Recherche des Templates correspondant au nœuds racine (indentification = "/").
  3. Enrichir la liste avec les nœuds fils et essayer d'appliquer récursivement les Templates correspondants à ces nœuds.
- **Exécution des Templates**
  - Ecriture sur la sortie
  - Mise à jour de la liste

Référencement de la feuille XSLT dans XML

```
<?xml version = "1.0"   ?>
<?xml-stylesheet  type="text/xsl"  href= "document.xsl"?>
…… la suite du document XML à transformer
```

Le type de la feuille associée

Chemin jusqu'à la feuille XSLT

Feuilles de style XSL/XSLT

Est un **document XML de type spécifique** .xsl ou .xslt qui consiste en :

- **Prologue** spécifique
  - <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

Le namespace des balises des feuilles de style XSLT

+ éléments caractérisant la feuille de style

  - <xsl:include href="path_file.xsl"/> ➔ inclure des règles d'une autre XSL
    - Elles ont la même priorité. En cas de conflit de règles c'est l'ordre d'apparition dans le document qui prime
  - <xsl:import href="path_file.xsl"/> ➔ importer des règles d'une autre XSL
    - Même chose que l'include à l'exception que les règles importées sont moins prioritaires que les règles du xslt qui importe
  - **<xsl:output method="xml|html|text|nom">** ➔ indique le format de sortie du document résultat

  Cet élément est à placer immédiatement après l'élément *<xsl:stylesheet>*

- **Templates**

Templates XSLT : Structure

<xsl:template

match = *pattern* *(chemin XPath sélectionne le nœud auquel s'applique ce template)*

name = *qname* *(nom du template – pour l'appeler à partir d'autres templates)*

priority= *number* *(priorité)*

mode = *qname* *(mode pour les traitements alternatifs selon appelant des modes spécifiques)*

<!-- contenu -->

</xsl:template>

Exemples de Template

- Un **document XML**

  This is an <emph> important </emph> point

- Un **template** comme une feuille de style

```
<xsl:template match="emph">
  <fo:inline-sequence font-weigth="bold">
        <xsl:apply-templates/>
  </fo:inline-sequence>
</xsl:template>
```

Appeler les Templates, si elles existent, correspondant aux balises filles de « emph »

- Le **résultat**

  This is an **important** point

Contenu d’un template

- Le contenu représente les éléments écrits sur la sortie
- Si un template contient du texte, ce texte sera écrit sur la sortie.

Ex :

```
<xsl:template mtach=“/”>
    Feuille de style remplaçant tout par ce texte ?!!!
</xsl:template>
```

Contenu d'un template

- Si un template contient **des éléments XML ainsi que des attributs XML**, ceux-ci seront écrit sur la sortie.

  Ex :

```
<xsl:template match="/">
        <HTML>
                <HEAD>…
                </HEAD>
        <HTML>
</xsl:template>
```

  } généré en sortie

- Il est néanmoins conseillé d'utiliser les **fonctions de création** de nœuds XML

Fonctions de création de nœuds XML

- Création d'un élément (balise)
  - <xsl:element name = *qname*> … </xsl:element>
- Création d'un attribut
  - <xsl:attribute name = *qname*> *valeur* </xsl:attribute>
- Création de texte
  - <xsl:text > *texte* </xsl:text >
- Création d'instructions de traitement
  - <xsl:processing-instruction name=*qname*>

    <?nom_instruction attributs="valeur"?> </xsl:processing-instruction>
- Création de commentaires
  - <xsl-comment> *text* </xsl-comment> s

Exemple : cataloge.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet  type="text/xsl"  href= "./cdcatalog.xsl"?>
<catalog>
    <cd>
        <title>Empire Burlesque</title>
        <artist>Bob Dylan</artist>
        <country>USA</country>
        <company>Columbia</company>
        <year>1985</year>
    </cd>
    <cd>
        <title>Hide your heart</title>
        <artist>Bonnie Tyler</artist>
        <country>UK</country>
        <company>CBS Records</company>
        <year>1988</year>
    </cd>
</catalog>
```

L'élément <xsl:template> : Exemple

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
  <body>
     <h2>My CD Collection</h2>
     <table border="1">
            <tr bgcolor="#9acd32">
                    <th>Title</th> <th>Artist</th>
            </tr>
            <tr> <td>.</td> <td>.</td> </tr>
     </table>
  </body>
 </html>
 </xsl:template>
</xsl:stylesheet>
```

Résultat

Fonction xsl:value-of

- La fonction ***<xsl:value-of>*** permet d'extraire la valeur d'un noeud sélectionné dans l'arborescence d'un document XML

  - syntaxe

    <xsl:value-of select = *path* >

Ex:

*extraire la valeur de l'attribut **name** de la balise **person***

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

L'élément <xsl:value-of>: Exemple

- Utilisé pour extraire la valeur du noeud sélectionné
- Exemple

```
<xsl:template match="/">
<html>
 <body>
    <h2>My CD Collection</h2>
      <table border="1">
          <tr bgcolor="#9acd32">
              <th>Title</th> <th>Artist</th>
          </tr>
          <tr>
              <td><xsl:value-of select="catalog/cd/title"/></td>
              <td><xsl:value-of select="catalog/cd/artist"/></td>
          </tr>
          </table>
 </body>
</html>
</xsl:template>
```

Résultat

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan

Expression Xpath

Traitement itératif

- La fonction ***<xsl:for-each>*** permet d'appliquer des règles de style sur chaque noeud identique d'un *template*.

```
<xsl:for-each select = path>
    Contenu
</xsl:for-each>
```

Ex:

```
<xsl:template match="/">
    <xsl:for-each select="eleve"> ... </xsl:for-each>
<xsl:template match="/">
```

L'élément <xsl:for-each> : Exemple

- Permet de boucler sur les fils d'un noeud du document

```
<xsl:template match="/">
<html>
  <body>
     <h2>My CD Collection</h2>
       <table border="1">
            <tr bgcolor="#9acd32">
                <th>Title</th> <th>Artist</th>
            </tr>
            <xsl:for-each select="catalog/cd">
            <tr>
               <td><xsl:value-of select="title"/></td>
                        <td><xsl:value-of select="artist"/></td>
            </tr>
            </xsl:for-each>
            </table>
  </body> </html>
</xsl:template>
```

Résultat

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary More
Eros	Eros Ramazzotti
One night only	Bee Gees
Maggie May	Rod Stewart
Romanza	Andrea Bocelli
When a man loves a woman	Percy Sledge
Black angel	Savage Rose
For the good times	Kenny Rogers
Big Willie style	Will Smith
Tupelo Honey	Van Morrison
The very best of	Cat Stevens
Stop	Sam Brown
Bridge of Spies	T'Pau
Private Dancer	Tina Turner
Midt om natten	Kim Larsen
Pavarotti Gala Concert	Luciano Pavarotti
The dock of the bay	Otis Redding
Picture book	Simply Red
Red	The Communards
Unchain my heart	Joe Cocker

L' élément <xsl:sort> : Exemple

Example

```
<xsl:for-each select="catalog/cd">
  <xsl:sort select="artist"/>
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
</xsl:for-each>
```

Result

My CD Collection

Title	Artist
Romanza	Andrea Bocelli
One night only	Bee Gees
Empire Burlesque	Bob Dylan
Hide your heart	Bonnie Tyler
The very best of	Cat Stevens
Greatest Hits	Dolly Parton
Sylvias Mother	Dr.Hook
Eros	Eros Ramazzotti
Still got the blues	Gary Moore
Unchain my heart	Joe Cocker
Soulsville	Jorn Hoel
For the good times	Kenny Rogers
Midt om natten	Kim Larsen
Pavarotti Gala Concert	Luciano Pavarotti
1999 Grammy Nominees	Many
The dock of the bay	Otis Redding
When a man loves a woman	Percy Sledge
Maggie May	Rod Stewart
Black angel	Savage Rose
Picture book	Simply Red
Red	The Communards
Private Dancer	Tina Turner
Tupelo Honey	Van Morrison
Big Willie style	Will Smith

Traitement conditionnel

- XSLT propose deux fonctions pour les traitements conditionnels
  - <xsl:if test = *boolean-exp*>
    Contenu
    </xsl:if>
  - <xsl:choose>
    <xsl:when test=*boolean-expr*> … </xsl:when>
    <xsl:when test=*boolean-expr*> … </xsl:when>
    …
    <xsl:otherwise> … </xsl:otherwise>

    </xsl:choose>

L'élément <xsl:if> : Exemple

- Permet d'exprimer des conditions
- **Syntaxe**

  <xsl:if test="*expression*">

  ... ...alors faire…

  </xsl:if>

Example

```
<xsl:for-each select="catalog/cd">
 <xsl:if test="price &gt; 10">
   <tr>
     <td><xsl:value-of select="title"/></td>
     <td><xsl:value-of select="artist"/></td>
   </tr>
 </xsl:if>
</xsl:for-each>
```

Résultat

My CD Collection

Title	Artist
Empire Burlesque	Bob Dylan
Still got the blues	Gary Moore
One night only	Bee Gees
Romanza	Andrea Bocelli
Black Angel	Savage Rose
1999 Grammy Nominees	Many

Fonction xsl:apply-templates

- La fonction ***<apply-templates>*** permet d'appliquer les templates d'une feuille de style sur les fils du noeud courant et les noeuds textuels.

<xsl:apply-templates
select = *path* ➔ *traiter les noeuds sélectionnés par le chemin* ***path***
mode = *name* > ➔ *appliquer les templates dont le mode est* ***name****.*
<!- - contenu -->
</xsl:apply-templates>

Ex :

<xsl:template match="\">
<xsl:apply-templates/> ➔ *appliquer tous les templates*
</xsl:template>

L'élément <xsl:apply-templates>: Exemple

```
<xsl:template match="/">
    <html>
        <body>
            <h2>My CD Collection</h2>
             <xsl:apply-templates/>
        </body>
    </html>
</xsl:template>

<xsl:template match="cd">
<p> <xsl:apply-templates select="title"/>
    <xsl:apply-templates select="artist"/> </p>
</xsl:template>

<xsl:template match="title">
    Title: <span style="color:#ff0000">
        <xsl:value-of select="."/></span> <br />
</xsl:template>

<xsl:template match="artist">
    Artist: <span style="color:#00ff00">
        <xsl:value-of select="."/></span> <br />
</xsl:template>
```

Résultat

My CD Collection

Title: Empire Burlesque
Artist: Bob Dylan

Title: Hide your heart
Artist: Bonnie Tyler

Title: Greatest Hits
Artist: Dolly Parton

Title: Still got the blues
Artist: Gary Moore

Title: Eros
Artist: Eros Ramazzotti

Title: One night only
Artist: Bee Gees

Title: Sylvias Mother
Artist: Dr.Hook

Fonction xsl:copy-of et xsl:copy

- La fonction ***<xsl:copy-of>*** est utilisée afin de recopier le nœud courant (et son arborescence) dans l'arborescence du document XML résultant. **<xsl:copy>** copiera juste le nœud en question (sans l' arborescence).
  - syntaxe

```
<xsl:copy-of select = path> </xsl:copy>
```

Ex:

```
<xsl:template match="/">
        <xsl:copy-of select="."/>
</xsl:template>
```

Création de variables

- La fonction **<*xsl:variable*>** permet de déclarer une variable dans une feuille XSL. Les variables sont des nœuds de tout type: balises, attribut…

  <xsl:varaible name=*qname* select=*path*>

- Déclarée dans un template, la variable a une portée locale.
- Une variable est référencée en utilisant son nom précédé de **$**.

Ex:

Déclaration : <*xsl:variable name=" num " select="@numero"/>*

Utilisation : <*xsl:apply-templates select="categorie/logiciel{$num}"*>

Autres fonctions XSLT

- Autres fonctions XSLT
  - Opérations sur les nombres
  - Opérations sur les chaînes de caractères
  - Envoi de messages
  - …

Utilisation des modes de Templates

- Modes : règles de traitement alternatives.
  - **déclaration**

    <xsl:template match="fils" mode="m1">

    <xsl:template match="fils" mode="m2">

    <xsl:template match="fils" mode="m3">

    …
  - **utilisation**

    <xsl:apply-templates mode="m2">

Invocation de Template

- L'utilisation de l'attribut **name** de l'élément **xsl:template** permet d'invoquer le template.
- Un template nommé peut ne pas avoir d'attribut **match**
- Un template nommé peut être invoqué par l'élément

<xsl:call-template name=*qname*>

Ex:

- Déclaration :

```
<xsl:template name="cellule">
        .....
</xsl:template>
```

- Invocation :

```
<xsl:call-template name="cellule"/>
```

Exemple : cataloge.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet  type="text/xsl"  href= "./cdcatalog.xsl"?>
<catalog>
    <cd>
        <title>Empire Burlesque</title>
        <artist>Bob Dylan</artist>
        <country>USA</country>
        <company>Columbia</company>
        <year>1985</year>
    </cd>
    <cd>
        <title>Hide your heart</title>
        <artist>Bonnie Tyler</artist>
        <country>UK</country>
        <company>CBS Records</company>
        <year>1988</year>
    </cd>
</catalog>
```

Exemple : cdcataloge.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
          <html> <body>
                    <h2>My CD Collection</h2>
                    <xsl:apply-templates/>
           </body> </html>
 </xsl:template>

 <xsl:template match="cd">
  <p> <xsl:apply-templates select="title"/> <xsl:apply-templates select="artist"/> </p>
 </xsl:template>

 <xsl:template match="title">
   Title: <span style="color:#ff0000"> <xsl:value-of select="."/></span> <br />
 </xsl:template>

 <xsl:template match="artist">
   Artist: <span style="color:#00ff00"><xsl:value-of select="."/></span><br />
 </xsl:template>

</xsl:stylesheet>
```

Exemple : le résultat

My CD Collection

Title: Empire Burlesque

Artist: Bob Dylan

Title: Hide your heart

Artist: Bonnie Tyler

Moteurs XSLT

- IE 5.0 (Microsoft)
  - Premier navigateur à proposer un moteur de transformation XSLT. Cependant, le moteur XSLT de IE5 n'est pas entièrement conforme au standard (mais pour IE6,7, 8 ...oui)
- Xalang (Apache)
  - Moteur de transformation proposé par le groupe Apache. Entièrement compatible avec le standard. Unité autonome.
- Turbine (Apache)

Exercice

```
<A>
    <B>
        <E>4</E>
        <E>3</E>
    </B>
    <B>
        <E>1</E>
    </B>
    <C>
        <F>a</F>
    </C>
    <E>2</E>
</A>
```

- xml1.xml

```
<A>
    <B>
        <E>4</E>
    </B>
    <B/>
    <C>
        <F>a</F>
    </C>
    <C>
        <F>b</F>
    </C>
</A>
```

xml2.xml

Donnez le contenu de la feuille de style XSLT qui a servit à produire xml2.xml à partir de xml1.xml.

Une solution:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match=’/’>
      <xsl:apply-templates select=’A’/>
</xsl:template>
<xsl:template match=’A’>
   <A>
     <xsl:apply-templates select=’B[1]’/>
    <B></B>
     <xsl:apply-templates select=’C’/>
    </A>
</xsl:template>

<xsl:template match=’B’>
       <B>
       <E> <xsl:value-of select=’E’/> </E>
       </B>
</xsl:template>

<xsl:template match=’C’>
       <C>
       <F> <xsl:value-of select=’F’/> </F>
       </C>
       <C>
       <F>b</F>
       </C>
</xsl:template>
</xsl:stylesheet>
```

Exercice

- Soit le document XML suivant

```
<AAA>
  <BBB/>
  <BCD/>
  <CCC>
    <BBB nom='titi' />
    <DDD nom='toto' />
  </CCC>
</AAA>
```

- Ecrire une feuille XSLT qui transforme ce document en un autre ne contenant que les balises AAA, BBB, CCC
- Ecrire une feuille XSLT qui copie le contenu de ce document dans un autre en inversant les valeurs des attributs *nom* des balises BBB et DDD
- Quelles sont vos critiques sur XSLT ?

Conclusion

- Langage de transformation de document XML
- Syntaxe XML (relativement simple)
- Fonctionnement Pattern/Matching
- En principe utiliser pour construire un document XSL
- Transformation Client ou Serveur
- Recommandation W3C Novembre 1999

Principes de base XSL/FO

Spécification de XSL
url: http://www.w3.org/TR/xsl/

But
- Formatage d'un document XML
- Qualité d'affichage de haut niveau (équivalent à celle d'un bon traitement de texte)
- Adaptation aux média (browser, imprimante, ...)

Usages
1. XML + XSLT => XML + XSL/FO -> format imprimable / affichable

Statut
- XSL/FO est une recommendation du W3C depuis 2001
- XSL-FO ne marche pas encore dans certains navigateurs habituels (mais il existe de viewers)
- Il existe plusieurs outils "server-side" (Cocoon, Axkit) qui incluent un processeur FO.
- Des éditeurs XML comme **Oxygen** peuvent lancer des processeurs FO comme FOP

url: **http://xmlgraphics.apache.org/fop/ (Processur XSL/FO de Apache)**

fo:root

La racine d'un document formaté est un élément : fo:root

Un document formaté commence donc par :

<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">

fo:root

L'élément fo:root contient :

- un élément : fo:layout-master-set
- 0 ou 1 élément : fo:declarations
- 1 ou plus élément : fo:page-sequence

Squelette XSLT vers XSL-FO de base

Voici la squelette XSLT vers XSL-FO de base

```
<xsl:template match="page">
<fo:root>
    <fo:layout-master-set>
    <!-- Definition of a single master page. It is simple (no headers etc.) -->
        <fo:simple-page-master master-name="first" >
            <!-- required element body -->
            <fo:region-body/>
        </fo:simple-page-master>
    </fo:layout-master-set>
    <!-- Definition of a page sequence -->
    <fo:page-sequence master-reference="first">
        <fo:flow flow-name="xsl-region-body" font-size="14pt" line-height="14pt">
          <xsl:apply-templates/>
        </fo:flow>
    </fo:page-sequence>
</fo:root>
</xsl:template>
```

Le traditionnel exemple: Bonjour Le Monde

```
<?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 master-name="LetterPage" page-width="8.5in“ page-height="11in" >
            <fo:region-body region-name="PageBody" margin="0.7in"/>
        </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-reference="LetterPage">
        <fo:flow flow-name="PageBody">
            <fo:block>Hello World</fo:block>
        </fo:flow>
    </fo:page-sequence>
</fo:root>
```

Resultat => Hello World

fo:layout-master-set

L'élément fo:layout-master-set permet de définir la mise en page du document

Il contient 1 ou plus élément

- fo:simple-page-master

ou

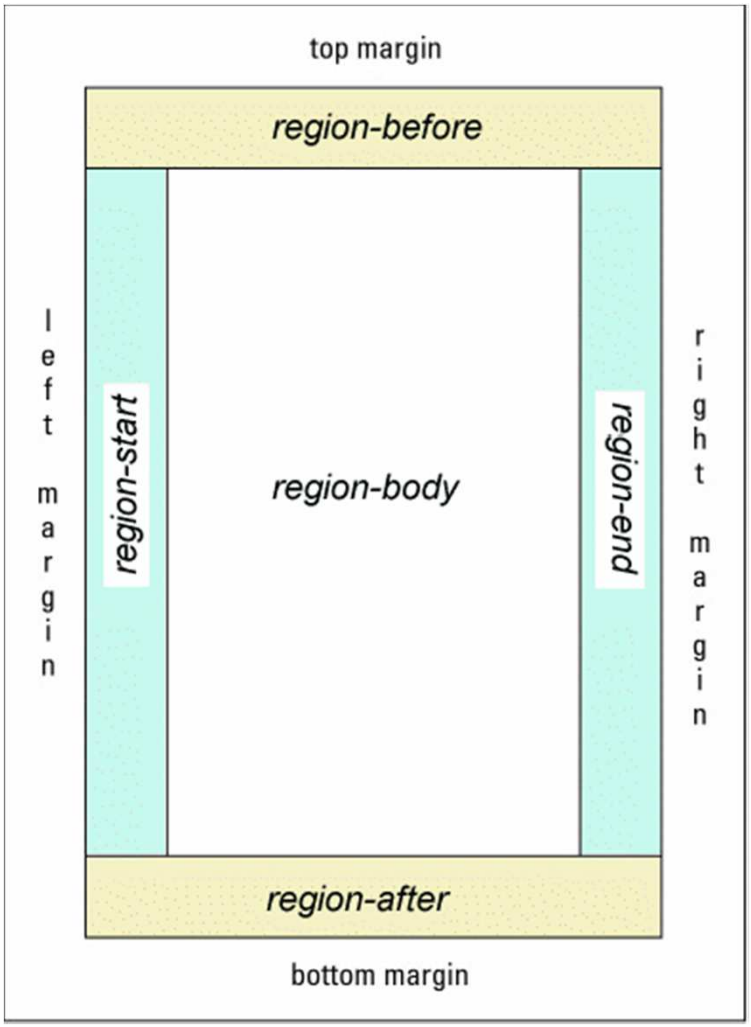
- fo:page-sequence-master

fo:simple-page-master

Définit la mise en page

```
<fo:simple-page-master master-name="simple"
    page-height="29.7cm"
    page-width="21cm"
    margin-top="1cm"
    margin-bottom="2cm" margin-left="2.5cm" margin-
    right="2.5cm">
      <fo:region-body margin-top="3cm"/>
      <fo:region-before extent="3cm"/>
      <fo:region-after extent="1.5cm"/>
</fo:simple-page-master>
```

La mise en page: référence



fo:page-sequence

L'élément fo:page-sequence permet de définir le contenu des pages.

Il contient :

0 ou 1 élément fo:title

0 ou plus élément fo:static-content

1 élément fo:flow

fo:flow

L'élément fo:flow permet de définir le flot de données contenu dans les pages.

Il contient des éléments de type block :

fo:block

fo:block-container

fo:table-and-caption

fo:table

fo:list-block

fo:block

L'élément fo:block est l'élément de base du formatage de paragraphe.

Il peut contenir :

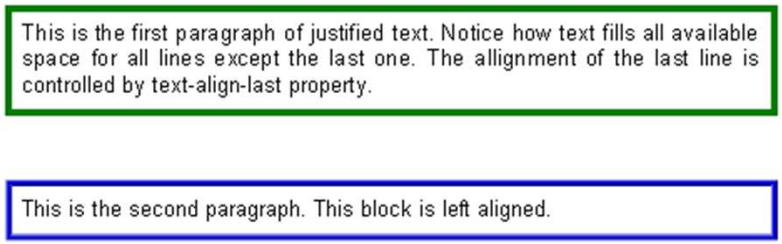
du texte

des éléments de type block

des éléments de type ligne

Les paragraphes: exemple

```
<?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 master-name="LetterPage"  page-width="6in" page-height="5in">
        <fo:region-body region-name="PageBody" margin="0.7in"/>
    </fo:simple-page-master>
</fo:layout-master-set>
<fo:page-sequence master-reference="LetterPage">
    <fo:flow flow-name="PageBody" font-family="Arial" font-size="12pt" >
    <fo:block text-align="justify" space-after="0.5cm" border="0.5pt solid green" >  C'est le
        premier paragraphe du texte justifié. Remarquez comment le texte remplit tout l'espace
        disponible. La bordure environnante est de 0.5 points de large, couleur verte et pleine.
        Ce paragraphe a un espace-après égale à 0.5 centimètres.
    </fo:block>
    <fo:block text-align="justify" space-before="2cm" border="0.5pt dotted red" >
    C'est le deuxième paragraphe du texte justifié. Cette fois la bordure est pointillée et rouge.
        Ce paragraphe a un espace-avant égale à 2 centimètres.
    </fo:block>
    </fo:flow>
</fo:page-sequence>
</fo:root>
```



XSL FO

- La présentation complète de XSL Format est loin d'être terminée!
- Le langage offre beaucoup de possibilités
- N’hésitez pas à consulter le standard