



# XML (eXtensible Markup Language)

Ibrahima FALL

[Ibrahima.Fall@esp.sn](mailto:Ibrahima.Fall@esp.sn)

Département Génie Informatique (DGI), Ecole Supérieure Polytechnique (ESP)

Université Cheikh Anta Diop (UCAD) de Dakar

BP 5085 Dakar-Fann, Sénégal

# PLAN

1. Historique
2. Syntaxe
3. Définition et Validation
  - DTD, W3 XML Schema
4. Feuilles de style
  - CSS, XSLT, XPath, XSLFO
5. APIs XML
6. Exemples d'applications XML
  - Apache Ant
  - RDF
  - ...

# EVALUATION

- 50% Contrôle continu
  - 1 TP noté
  - 1 projet
- 50% Examen
  - Sur table.

# ORGANISATION

- 32h de CM/TD/TP
- Au nom du respect mutuel **[J'INSISTE]**
  - ☐ Être en classe à l'heure prévue par l'emploi du temps
    - Prière de ne pas rentrer en classe une fois le cours commencé
  - ☐ Se garder de manger/boire en classe
  - ☐ Ne pas être l'auteur de dérangements sonores (portables, bruits de machines, etc.)
  - ☐ Se passer de son téléphone et d'Internet



Chapitre 1

# Historique de XML

De SGML à ... XHTML.

# 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/eXternal Data Representation
- ☐ etc.

# 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**

# Qu'est que SGML?

- Une norme internationale :
  - Standard Generalized Markup Language
  - ISO 8879 – 1989
  - Premier essai normalisé pour les documents électroniques
- Un métalangage de balisage de documents
  - Lisible par l'être humain et traitable par une machine
  - Permet de définir des langages de balisage
- Les documents sont balisés conformément à la grammaire (la DTD)
  - Instances de DTD
  - Permet un balisage sémantique du fond



# 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 forcément compatible WEB !**

# Qu'est que HTML?

- Version allégée de SGML dont il est une dérivée: existence de DTD HTML
- Proposé par le W3C comme format de documents sur le Web
- Langage simple avec des balises standardisées permettant la mise en forme d'un texte
- Standard du développement Web, très facile à apprendre et à utiliser
- Très industrialisé
  - Standard reconnu par tous les navigateurs.
  - Plusieurs logiciels WYSIWYG (Front Page, Dreamweather) et outils de publication de contenu (CMS) (Spip, eZPublish, PHPNuke)

# 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 !
- 👎 **Limité à la structuration de pages web**
  - structure (*ordre des balises*),
  - données (*type, valeur*),
  - sémantique
- 👎 ...

# HTML-SGML (résumé critique)

## ■ SGML

- Langage de la GED **très puissant** mais très complexe

## ■ HTML

- Instance **simple** de SGML
- **Adapté à la présentation**
- Inadapté à l'échange entre programmes

**Les concepteurs de XML ont cherché à exploiter les avantages de ces 2 technologies tout en se débarrassant de leurs inconvénients**

# XML

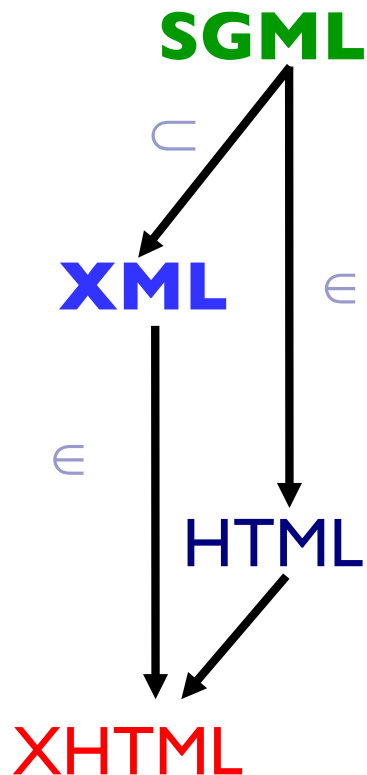
## Définition intuitive d'XML:

- **XML :**  $\left\{ \begin{array}{l} \text{→ - variante de **HTML généralisé** !} \\ \text{ } \\ \text{→ - **sous-ensemble de SGML** !} \end{array} \right.$ 
  - (compatibilité WEB, lisibilité, syntaxe)*
  - (flexibilité, rigueur)*
- langage à balises configurables
- pour la représentation hiérarchique de données,
- <http://www.w3.org/XML/>

# XML

## Exemple de document

```
<Etudiant>
  <Age>22</Age>
  <Region> <Nom>Nord</Nom>
  <Capitale>Saint-Louis</Capitale>
</Region>
  <Entrée>1995</Entrée>
  <Moyenne>15.7</Moyenne>
  <ValeurPrix Unite = "FCFA"> 105K
</ValeurPrix>
</Etudiant>
```



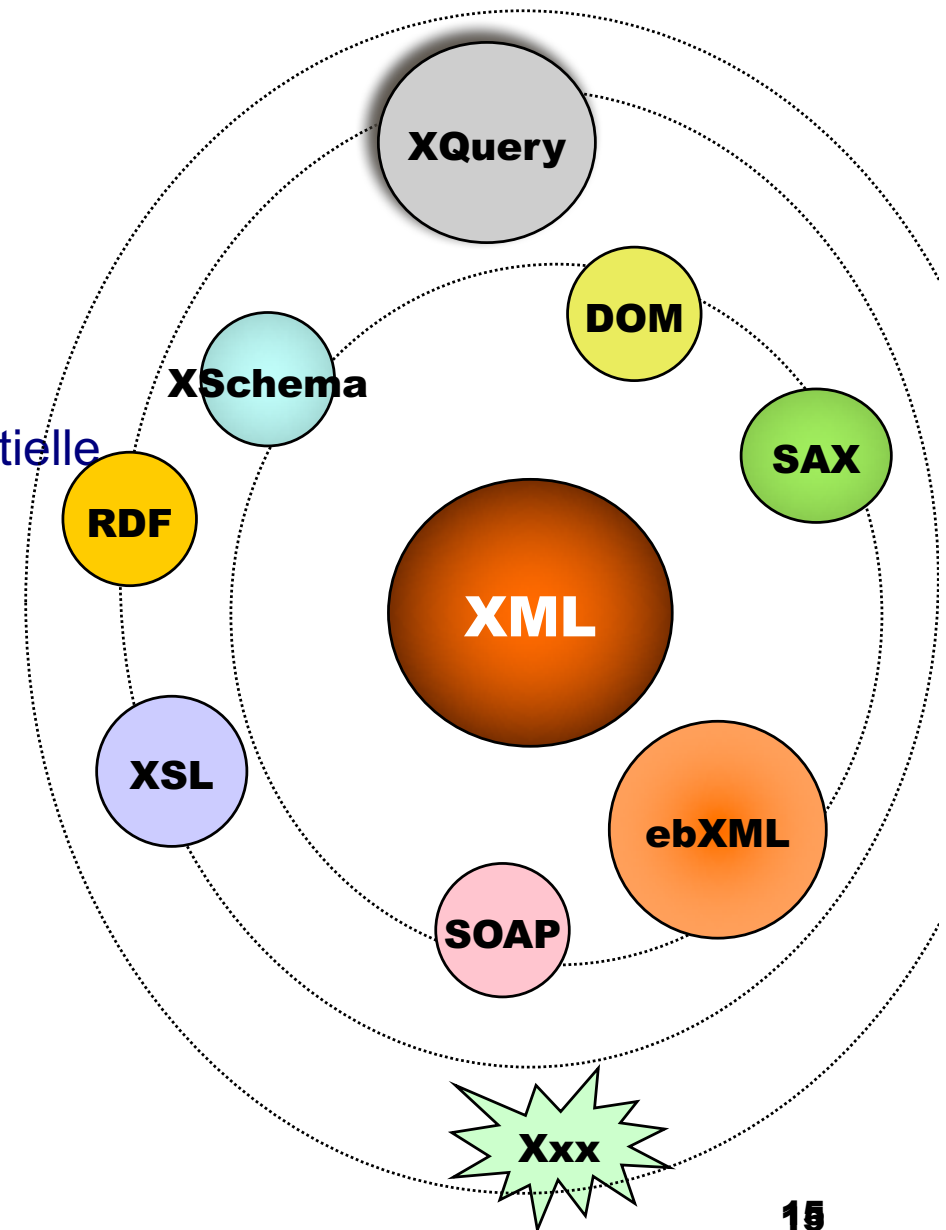
*Les utilisateurs peuvent définir leurs propres tags*

*Il est possible d'imposer une grammaire spécifique (DTD, Schéma)*

*Les tags indiquent la signification des sections marquées*

# XML: une galaxie de standards

- **XSchema**
  - Schémas de documents
- **XSL**
  - Feuilles de styles
- **SAX**
  - API de programmation événementielle
- **DOM**
  - API de programmation objet
- **SOAP**
  - Protocole Web Services
- **RDF**
  - Description de ressources Web
- **ebXML**
  - Standards de e-Commerce
- **Xxx**
  - Standards par métiers ...



# XML: un standard de fait

- Un standard d'échange
  - Lisible : texte balisé avec marquage
  - Clair : séparation du fond et de la forme
  - Extensible : supporte les évolutions applicatives
  - Sécurisé : pare-feu, encryption, signature
- Développé par le W3C
  - Pour le Web (Internet, Intranet)
  - S'étend à l'entreprise et ses partenaires
- Supporté par les grands constructeurs
  - IBM, Microsoft .net, SUN, BEA, etc.
  - Des outils génériques et ouverts
- Transversale à l'entreprise
  - Échanges de données, Bases de données, ...
  - Bureautique, Intégration eBusiness, ...
  - GED, Sites Web, ...



# XHTML

## ■ eXtensible HTML

- ☐ Reformulation de HTML 4 en tant qu'application XML 1.0
- ☐ Plusieurs avantages
- ☐ Prochaine (actuelle?) étape de l'évolution d'Internet

# Un mot sur le W3C

- Word Wide Web Consortium
- Fondé en 1994
- Consortium industriel international accueilli par différents sites
  - MIT/LCS aux Etats-Unis
  - INRIA en Europe
  - Keio University au Japon
- 322 membres (universitaires et industriels) en mi-janvier 2011
  - <http://www.w3.org/Consortium/Member/List>



## Chapitre 2

# La syntaxe XML

Structure, Eléments de  
documents XML, XML Bien  
Formé, Validité.

# Structure de documents XML

## ■ Prologue :

- Rôle équivalent au **<HEAD> HTML**,

- **Meta-Information** :
  - **instructions** de traitement
  - **commentaires**  
*(non interprétables par le parseur)*

## ■ Corps :

- Rôle équivalent au **<BODY> HTML**

- **Les données formatées**:  
*(structure arborescente)*
  - **Balises** d'encadrement
  - **Attributs associées** aux balises
  - **Données encadrées** par les balises

# Exemple XML : *Une lettre*

## PROLOGUE

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

*document XML  
instruction de traitement*

*document autonome*

*jeu de caractères utilisé  
(latin)*

*balise début*

## CORPS

```
<lettre>
  <lieu> Somewhere in space</lieu>
  <expéditeur> I. Fall</expéditeur>
  <destinataire> M. Camara</destinataire>
  <introduction> Dear folk, </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)

*Jeu de caractères utilisé (**Facultatif**) (Ici ISO-8859-1: jeu LATIN, pour prendre en compte les accents français )*

*Document XML  
1.0*

*Ceci est un document XML non autonome  
(il utilise une définition externe) (**Facultatif**)*

*(**Obligatoire**)*

```
<?xml version="1.0" standalone="no" encoding="ISO-8859-1" ?>  
<?xml-stylesheet type="text/xsl" ?>  
<!DOCTYPE liste_CD SYSTEM "CDs.dtd">
```

*[Un commentaire spécial !]  
(il définit le type de document XML)  
(**Facultatif**)*

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

*Autre instruction de traitement  
(**Facultatif**)*

# Corps d'un document XML (Exemple)

```

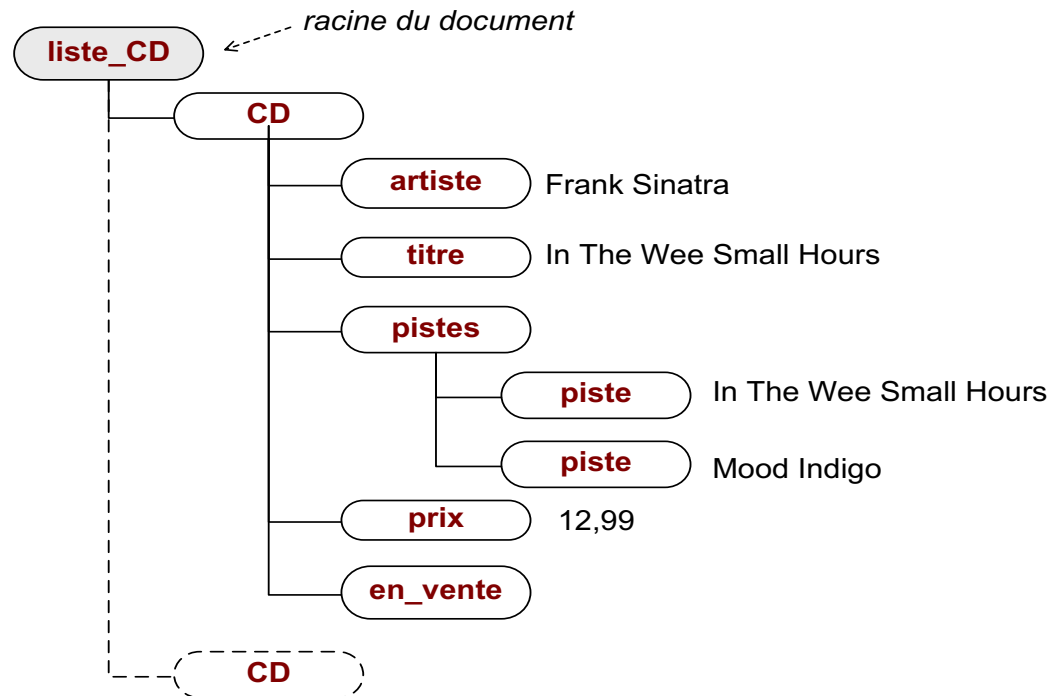
1
<liste_CD>
  <CD>
    <artiste type="individual">Frank Sinatra</artiste>
    <titre no_pistes="4">In The Wee Small Hours</titre>
    <pistes>
      <piste>In The Wee Small Hours</piste>
      <piste>Mood Indigo</piste>
    </pistes>
    <prix mannaie="euro" paiement="CB">12.99</prix>
    <en_vente/>
  </CD>
  <CD> ..... </CD>
  .....
</liste_CD>
2
  
```

Diagram illustrating the body of an XML document with annotations:

- 1: Start of the root element `<liste_CD>`.
- 2: End of the root element `</liste_CD>`.
- 3: Start of the `<CD>` element.
- 4: End of the `<en_vente/>` element.
- 5: Attribute `type="individual"` of the `<artiste>` element.
- 6: End of the `<artiste>` element.
- 7: Attribute `paiement="CB"` of the `<prix>` element.
- 8: Attribute `mannaie="euro"` of the `<prix>` element.
- 9: End of the `<CD>` element.

# 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 précédent*)
- 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) contiennent des **attributs**

# Structure des documents XML : Synthèse

- Un document XML : Prologue + Corps  
(un arbre de balises)

- Balises du prologue :

`<?nom_balise_traitement .... ?>`

*Instruction de traitement*

➤ *Transmis directement à une application spécifique*

`<!DOCTYPE .....>`

.....

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

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

- ou **uniques**

`<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 précedence !
- 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"/>
```

# Caractères de documents XML

- Le jeu de caractère d'un document XML est l'alphabet international Unicode
  - Permet pratiquement de représenter toutes les langues de la planète!
- Autres normes
  - UTF-8, UTF-16, ISO-8859-1
  - Voir <http://www.w3.org/International/O-charset>
- Parseur XML
  - Logiciel d'analyse d'un document XML
  - Doit au moins supporter les encodages définis par UTF-8 et UTF-16
- Les espaces
  - « », « \t », « \n », ... sont interprétés comme des espaces
- Le respect de la casse
  - Les identificateurs <Titre> et <titre> sont différents

# Les instructions de traitement

- Nous avons vu que le prologue pouvait être complété par des instructions de traitement destinées aux applications qui vont utiliser le document XML sous la forme
  - `<?Application instruction+ ?>`
- Par exemple pour permettre au navigateur de visualiser un document avec sa feuille de style XSL on utilise l'instruction
  - `<?xml-stylesheet type="text/xsl" ?>`
- Et pour lui permettre de visualiser un document avec sa feuille de style CSS on utilise l'instruction
  - `<?xml-stylesheet type="text/css" ?>`
- Enfin, l'exemple suivant demande au navigateur de visualiser un document avec la feuille de style CSS que nous lui indiquons
  - `<?xml-stylesheet type="text/css" href="./monCSS.css" ?>`

# Les références aux entités

- Une entité XML est une sorte de variable (en programmation classique)
  - Avec un nom et une valeur
  - Définie dans une DTD

- Exemple

...

`<exple>`

*Ceci est **&MonEntite;***

`</exple>`

...

- **&MonEntite;** est une référence vers l'entité **MonEntite** définie dans la DTD du document contenant cette section; le parseur va le remplacer par sa valeur textuelle, par exple «Une illustration d'une référence vers une entité»

# Les références aux caractères

- Certains caractères spéciaux sont réservés à la définition d'un document XML; ils ne peuvent donc pas être directement utilisés
  - D'autres ne sont pas accessibles au clavier
- Il est donc nécessaire de connaître la valeur de ces caractères dans l'alphabet Unicode, ou leur code numérique
- Exemples:
  - **&lt;**
    - Une référence au caractère « < »
  - **&gt;**
    - Une référence au caractère « > »
  - **&#233;**
    - Une référence au caractère « é »

# Les sections CDATA

- Introduites sous la forme `<![CDATA[ ... ]]>`
- Elles ne sont pas analysées par le parseur; les règles de syntaxe d'XML ne s'y appliquent donc pas
- Exemple
  - La chaîne du type suivant est donc bien correcte
    - Ceci est `<![CDATA[` un exemple de section CDATA dans `<?xml version=«1.0?>.>.>`



# 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>
```

## ■ Conséquences

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

# Règles syntaxiques du langage XML

- Présence d'un prologue
- Existence d'un seul élément racine
  - Encore appelé **élément document**
- Les attributs sont associés aux balises ouvrantes et respectent la syntaxe de définition ***attr="valeur"***
- Toute balise qui s'ouvre doit se refermer
  - Avec respect des règles d'imbrication
- Le respect des règles de construction des identificateurs est assurée
  - Formés de caractères alphanumériques sans espaces
  - Ne commencent pas par un chiffre

# 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**  
(référéncé 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.
- Conséquence
  - Le document XML peut être échangé ! (*format standardisé*)



## Exercice

- Proposez un document XML bien formé représentant un ensemble références bibliographiques
  - [On peut adapter l'exemple selon le profil des participants]
- Si vous voulez traiter ces références bibliographiques, de quoi auriez 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

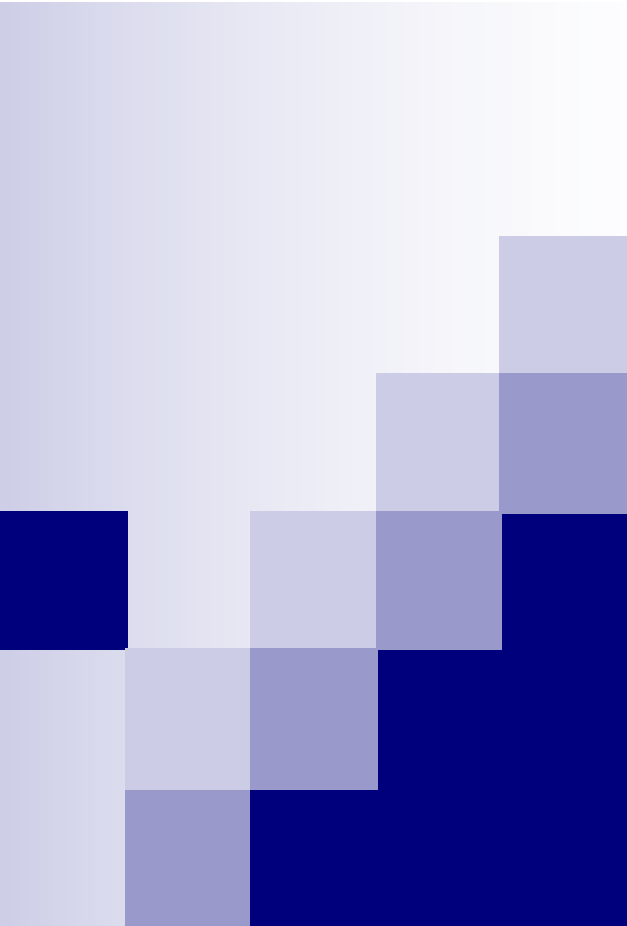
- Comment spécifie-t-on ce qui doit être affiché et comment ?

 **style** → CSS  
**transformations** → XSLT, DOM, XPath

.....

- L'ordre des attributs est-t-il une importance ?

→ non



Chapitre 3

# Définition et validation de documents XML

DTDs, W3 XML Schemas.

# Introduction

- Pour un document XML
  - Une DTD ou un schéma pour décrire les balises
  - Une feuille de style pour adapter le format aux besoins
    - Nous y reviendrons au chapitre suivant
- La DTD permet de définir son propre langage basé sur XML
  - Vocabulaire
  - Grammaire



# Document bien formé et valide

## ■ Document bien formé

- ☐ Respect des règles syntaxiques
- ☐ **Pas nécessairement conforme à une DTD ou schema**

## ■ Document valide

- ☐ Bien formé + conforme à une DTD (ou un schéma)

# DTD

- Document Type Definition
- Langage de modélisation de XML 1.0
- Permet de définir le «vocabulaire» et la structure qui seront utilisés dans le document XML
  - Définit un type de document par des spécifications précises
    - Permet de vérifier la validité d'un document
- Grammaire du langage dont les phrases sont des documents XML (instances)
- Peut être mise dans un fichier et être appelé dans le document XML
- Assure l'uniformité d'un ensemble de documents similaires
- Document non XML (Syntaxe héritée de SGML)
- La DTD n'est pas obligatoire
  - Un document qui fait référence à une DTD doit respecter ses spécifications

## Exemple de DTD

<!ELEMENT note(de, a, objet, description)>

<!ELEMENT de(#PCDATA)>

<!ELEMENT a(#PCDATA)>

<!ELEMENT objet(#PCDATA)>

<!ELEMENT description(#PCDATA)>

# Attributs et éléments

- `<!ELEMENT balise (contenu)`
  - Décrit une *balise* qui fera partie du vocabulaire
  - ex : `<!ELEMENT livre (auteur, editeur)>`
- `<!ATTLIST balise [attribut type #mode [valeur | valeur par défaut]]*`
  - Définit la liste d'attributs pour une balise déjà défini
  - ex :

<code>&lt;!ATTLIST</code>	<code>genre</code>	<code>CDATA</code>	<code>#REQUIRED</code>
	<code>ville</code>	<code>CDATA</code>	<code>#IMPLIED&gt;</code>
<code>&lt;!ATTLIST</code>			<code>editeur</code>
	<code>ville</code>	<code>CDATA</code>	<code>#FIXED "Paris"&gt;</code>

# Structuration des balises

## ■ Structuration du contenu d'une balise

- (a, b) séquence
  - (nom, prenom, rue, ville)
- (a|b) liste de choix
  - (oui|non)
- a? élément optionnel [0,1]
  - (nom, prenom?, rue, ville)
- a\* élément répétitif [0,N]
  - (produit\*, client)
- a+ élément répétitif [1,N]
  - (produit\*, vendeur+)

# Structuration des balises: exemples

- Structuration du contenu d'une balise
- Exemple 1: Élément MONTAGNE avec un ou plusieurs nom, et une hauteur optionnelle
  - *<!ELEMENT MONTAGNE(NOM+, HAUTEUR?, PAYS)>*
- Exemple 2: Élément MONTAGNE avec des sous-éléments à occurrence multiple
  - *<!ELEMENT MONTAGNE(NOM, HAUTEUR, PAYS)\*>*
- Exemple 3: Emboîtement de sous éléments
  - *<!ELEMENT MONTAGNE(NOM, HAUTEUR, (DEPARTEMENT|REGION|PAYS))>*

# Types (pour les éléments)

- #PCDATA

- ☐ Élément de texte sans descendants ni attributs contenant des caractères

- ANY

- ☐ Tout texte possible - pour le développement

- EMPTY

- ☐ Vide

# Types (pour les attributs)

- **CDATA**
  - Données brutes qui ne seront pas analysées
- **Enumération**
  - Liste de valeurs séparées par « | »
- **ID et IDREF/IDREFS**
  - Clé et référence (liste de références) pour les attributs
- **ENTITY/ENTITIES**
  - Nom (liste de noms) d'entités non XML déjà déclarées
- **NMTOKEN/NMTOKENS**
  - Mots clés (liste de mots clés)
- **NOTATION**
  - Notation (voir plus loin)



# DTD et documents XML

## ■ DTD interne

- Directement dans le document XML
- **<!DOCTYPE nom\_element\_document [**  
    *... Spécifications ICI ...*  
**]>**

## ■ DTD externe

- Déclarée séparément dans un autre fichier (.dtd);
- **<!DOCTYPE**                      **nom\_element\_document**  
    **SYSTEM|PUBLIC "URL" >**

# Exemple: DTD externe

docint.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>
```

```
<?xml version= "1.0"?>
<!DOCTYPE note SYSTEM "../docint.dtd " >
<doc>
...
</doc>
```

```
<?xml version= "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html>
...
</html>
```

# Exemple: 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>6 Millions FCA</PRIX>
  </VOITURES>
  .....
</CATALOGUE>

```

# Autre Exemple: (avec 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>
```

# Intérêt des DTD externes

- Modèle pour plusieurs documents
  - Partage des balises et structure
- 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">
...
```

# Les entités dans les DTD

- Modèle pour plusieurs documents
  - Partage des balises et structure
- Entités internes
  - Permet la réutilisation dans une DTD (entités paramètres)
    - `<!ENTITY %nom "definition">`
    - Utiliser dans la DTD par **%nom**;
  - Exemple
    - `<!ENTITY %sexe ("homme" | "femme")>`  
`<!ATTLIST auteur sexe %sexe; #REQUIRED>`
- Peuvent être externes
  - Pour les entités DTD (paramètres):
    - `<!ENTITY %regles SYSTEM "http://www.monsite.com/regles.dtd">`
    - Référencée par **%regles**;
  - Pour les entité générales
    - `<!ENTITY documentation SYSTEM "http://www.site.com/doc.xml">`
    - Référence dans un fichier XML **&documentation**;

# Comment utiliser les entités

## ■ Modularité

- ☐ Définir dans des entités séparées les parties réutilisables

## ■ Précédence

- ☐ Regrouper les déclarations d'entités en tête

## ■ Spécificité

- ☐ Éviter les DTD trop générales

## ■ Simplicité

- ☐ Découper les DTD trop complexes

# Les entités non-XML

- Une entité non-XML est un bloc d'information qui ne sera pas analysé par un parseur XML
  - Les données de ce bloc peuvent donc avoir un format quelconque et ne pas respecter les règles syntaxiques des documents XML
- Exemples
  - Les entités précédemment vues sont des entités XML
  - `<!ENTITY Inclusion0 SYSTEM "toto.xml">`
    - Est une entité XML
  - `<!ENTITY % Inclusion1 SYSTEM "toto.dtd">`
    - Est une entité XML
  - `<!ENTITY Inclusion2 SYSTEM "image.png">`
    - Est une entité **non XML**



# Les notations

- Les notations identifient par leur nom le format des entités non-XML ou d'un élément possédant un attribut de type NOTATION
- La déclaration d'une notation comprend
  - Le nom
    - Utilisé dans les autres déclarations (d'entités, d'attributs, etc.)
  - Un identifiant externe
    - Permettant au parseur d'identifier l'application qui doit traiter les données identifiées par la notation
- Une notation permet de déclarer une entité non-XML et d'y associer une application capable de traiter les données

# Les notations: exemple

- Déclaration de deux formats de données compressées avec les applications qui permettent de les traiter
  - `<!NOTATION gzip SYSTEM "gzip.exe">`
  - `<NOTATION compress SYSTEM "compress.exe" >`
- Déclaration d'une entité non-XML qui référence le fichier arch1.z au format compressé par gzip.exe
  - `<ENTITY arch1 SYSTEM "arch1.z" NDATA gzip>`
- Déclaration d'un type d'élément `<archive>`
  - `<!ELEMENT archive EMPTY>`
  - `<ATTLIST archive`  
Codage `NOTATION(gzip|compress)#REQUIRED`  
Contenu `ENTITY #IMPLIED>`
- Dans un document XML, on fait référence à l'entité **arch1** contenant les données à compresser
  - `<archive Codage= "gzip" Contenu= "arch1" >`

# Synthèse DTD

- Spécification de la structure du document
  - Déclaration de balisage : ELEMENT, ATTLIST, ENTITY;
  - Déclaration des éléments
    - Eléments 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)

# Insuffisance des DTD

- Tout doit être défini
  - Pas de modélisation partielle
- Pas de types de données
  - Difficile à interpréter
    - Syntaxe différente de celle des documents XML
  - Difficile à traduire en schéma objets
  - Pas d'héritage
  - Typage faibles
    - Pas de contraintes sur les données
- Propositions de compléments
  - XML-Schema du W3C



# Exercice

- Soit la structure arborescente suivante

- contenu

- introduction

- histoire

- etat\_actuel

- premiers pas

- un petit exemple

- Cette structure peut bien être représentée par un document XML valide par rapport à la DTD suivante. Créer ce document XML.

```
<!ELEMENT liste (point)*>
```

```
<!ELEMENT point (#PCDATA)>
```

```
<!ATTLIST point nom ID #REQUIRED point_parent IDREF  
#IMPLIED >
```



# Solution de l'exercice

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

<!DOCTYPE liste SYSTEM "liste.dtd">

<liste>
  <point nom="racine">contenu</point>
  <point nom="introduction" point_parent="racine">introduction</point>
  <point nom="histoire" point_parent="introduction">histoire</point>
  <point nom="aujourd_hui " point_parent="introduction">etat_actuel</point>
  <point nom="pas" point_parent="racine">premiers pas</point>
  <point nom="exemple" point_parent="pas">un petit exemple</point>
</liste>
```

# Objectifs des schémas

- Reprendre les acquis des DTD
  - Plus riche et complet que les DTD
- Permettre de typer les données
  - Eléments simples et complexes
  - Attributs simples
- Permettre de définir des contraintes
  - Occurrence obligatoire ou optionnelle
  - Cardinalités, références
- Permettre une modélisation partielle des documents
- Permettre une réutilisation de définitions existantes, avec les espaces de nommages
- ...

# W3 XML Schema

- Un schéma d'un document 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



# Définition d'un schema

- 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"?>  
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
    //...  
    //...  
</xsd: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>:Cq;c rvc      tytyt bqw</to>
```

```
  <from>Ibrahima</from>
```

```
  <heading>Reminder</heading>
```

```
  <body>Don't forget it this week-end!</body>
```

```
</note>
```

# Déclaration d'un élément simple

- Un élément simple contient des données dont le type est simple
  - Exemple: types de base en java
- 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>
```

- Autres exemples

- ```
<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

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

# Les types simples #2

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

# Les types simples #3

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

# Les types simples #4

<i>Type</i>	<i>Description</i>
<b>long</b>	<i>représente un nombre entier long dont l'intervalle est : {-9223372036854775808 - 9223372036854775807}</i>
<b>int</b>	<i>représente un nombre entier dont l'intervalle est : {-2147483648 - 2147483647}</i>
<b>short</b>	<i>représente un nombre entier court dont l'intervalle est {-32768 - 32767}</i>
<b>byte</b>	<i>représente un entier dont l'intervalle est {-128 - 127}</i>
<b>nonNegativeInteger</b>	<i>représente un nombre entier positif incluant le zéro</i>
<b>unsignedLong</b>	<i>représente un nombre entier</i>
<b>long</b>	<i>non-signé dont l'intervalle est {0 - 18446744073709551615}</i>

# Les types simples #5

<i>Type</i>	<i>Description</i>
<b>unsignedInt</b>	<i>représente un nombre entier non-signé dont l'intervalle est : {0 - 4294967295}</i>
<b>unsignedShort</b>	<i>représente un nombre entier court non-signé dont l'intervalle est : {0 - 65535}</i>
<b>unsignedByte</b>	<i>représente un nombre entier non-signé dont l'intervalle est {0 - 255}</i>
<b>positiveInteger</b>	<i>représente un nombre entier positif commençant à 1</i>



# Déclaration d'un attribut

- Tous les attributs sont de type simple
- Un attribut est défini selon la syntaxe suivante
  - `<xsd:attribute name = "....." type= "....." />`
- Exemple
  - `<xsd:attribute name="language" type="xsd:string"/>`
  - `<xsd:attribute name="country" type="xsd:NMTOKEN" fixed="FR"/>`
- Association d'un attribut à un élément
  - `<xsd:element ...> <xsd:attribute ...../> </xsd:element>`

# Déclaration d'un élément complexe

- Un élément complexe contient des données dont le type est complexe
  - Une structure, par exemple
- 3 compositeurs existe pour définir les 3 catégories essentielles de types complexes
  - **<sequence>**
    - Collection ordonnée d'éléments typés
  - **<all>**
    - Collection non ordonnée d'éléments typés
  - **<choice>**
    - Choix entre éléments typés

# Les types complexes

- Déclarer un élément complexe = définir son type + association du type à l'élément
- Deux façons 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="xs:string"/>  
      <xsd:element name="lastname" type="xs:string"/>  
    </xsd:sequence>  
  </xsd:complexType>  
</xsd:element>
```

# Les types complexes

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

Schéma XML correspondant au document précédent

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

- Cette seconde déclaration permet la réutilisation de types

- Exemple

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

# Le compositeur **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 compositeur **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 compositeur **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 #1

- Définition de sous-types par héritage de types simples 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="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 #2

## ■ Par restriction

- En utilisant des expressions régulières (patterns), ou des facettes on peut définir des contraintes sur des types simples

## ■ Exemple

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

# Restriction de types: les facettes

- maxLength, minLength
- maxInclusive, minInclusive
- pattern
- enumeration
- ...

# Exemple de construction d'un XML Schema

## ■ Document XML de référence

```
<?xml version="1.0" encoding="ISO-8850-1"?>
<book isbn="0836217462">
  <title> Being a Dog Is a Full-Time Job </title>
  <author>Charles M. Schulz</author>
  <character>
    <name>Snoopy</name>
    <friend-of>Peppermint Patty</friend-of>
    <since>1950-10-04</since>
    <qualification> extroverted beagle </qualification>
  </character>
  <character>
    <name>Peppermint Patty</name>
    <since>1966-08-22</since>
    <qualification>bold, brash and tomboyish</qualification>
  </character>
</book>
```

# L'élément `<schema>`

- Un XML Schema est un document XML
- L'élément `"schema"` ouvre un schéma W3C XML Schema
  - Il peut également contenir la définition de l'espace de nom cible et d'autres options dont nous verrons certaines dans la suite.

```
<?xml version="1.0" encoding="utf-8"?>  
  
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">  
  <!-- elements and attributes here -->  
  
</xsd:schema>
```

# 3 styles de définition de schéma

## ■ 1<sup>er</sup> style

- Suivre la structure du document XML de référence et définir chaque élément au moment où on le rencontre

## ■ 2ieme style

- Utiliser des éléments déjà défini pour construire, à la manière des DTDs, un catalogue d'éléments présents dans le document en précisant, pour chacun, la liste de ses attributs et éléments

## ■ 3ieme style

- Style intermédiaire
- Définir des types de données pouvant être simples ou complexes à utiliser ensuite ces types pour définir des éléments et attributs.

# 1er style de schéma

- Pour écrire un schéma, suivre la structure du document XML de référence et définir chaque élément au moment où on le rencontre. Utiliser les éléments suivants:
  - `<xsd:element>`
    - Pour définir un élément
  - `<complexType>`
    - Éléments ayant des attributs et des sous éléments
  - `<simpleType>`
    - Est alors réservé aux éléments et attributs sans sous-éléments ou attributs et ne contenant donc que des valeurs.
  - `<xsd:attribute>`
    - Pour définir les attributs de l'élément `<xsd:element>` dans lequel il est définit; les attributs devant toujours être déclarés après les éléments.
  - `<xsd:any>` et `<xsd:anyAttribute>`
- Un schéma XML pour les document XML de la famille du document référence pourrait être le schéma du slide suivant.

# 1er style de schéma: exemple

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:element name="book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="author" type="xsd:string"/>
        <xsd:element name="character" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="name" type="xsd:string"/>
              <xsd:element name="friend-of" type="xsd:string"
                minOccurs="0" maxOccurs="unbounded"/>
              <xsd:element name="since" type="xsd:date"/>
              <xsd:element name="qualification" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="isbn" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```



## 2eme style de schéma

- On utilise des éléments déjà définis pour construire un catalogue d'éléments présents dans le document
  - En précisant, pour chacun, la liste de ses attributs et éléments.
  - Comme dans les DTDs
- Pour cela utiliser des références à des éléments et attributs définis dans le champ de définition du référenceur
  - On crée ainsi une structure plate.
- L'attribut ref de element et attribute permet de jouer le rôle de référenceur.

## 2eme style de schéma: exemple #1

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <!-- definition of simple type elements -->
  <xsd:element name="title" type="xsd:string"/>
  <xsd:element name="author" type="xsd:string"/>
  <xsd:element name="name" type="xsd:string"/>
  <xsd:element name="friend-of" type="xsd:string"/>
  <xsd:element name="since" type="xsd:date"/>
  <xsd:element name="qualification" type="xsd:string"/>
  <!-- definition of attributes -->
  <xsd:attribute name="isbn" type="xsd:string"/>
  ....
```

## 2eme style de schéma: exemple #2

```
...
<!-- definition of complex type elements -->
  <xsd:element name="character">
    <xsd:complexType>
      <xsd:sequence>
        <!-- the simple type elements are referenced using the "ref" attribute -->
        <xsd:element ref="name"/>
        <!-- the definition of the cardinality is done when the elements are referenced -->
        <xsd:element ref="friend-of" minOccurs="0" maxOccurs="unbounded" />
        <xsd:element ref="since"/>
        <xsd:element ref="qualification"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
....
```

## 2eme style de schéma: exemple #3

```
...  
<xsd:element name="book">  
  <xsd:complexType>  
    <xsd:sequence>  
      <xsd:element ref="title"/>  
      <xsd:element ref="author"/>  
      <xsd:element ref="character" minOccurs="0" maxOccurs="unbounded"/>  
    </xsd:sequence>  
    <xsd:attribute ref="isbn"/>  
  </xsd:complexType>  
</xsd:element>  
</xsd:schema>
```

# 3eme style de schéma

- Utiliser des éléments déjà définis pour construire un catalogue d'éléments présents dans le document
  - Définir des types de données pouvant être simples ou complexes à utiliser ensuite ces types pour définir des éléments et attributs.
    - Donner un nom aux éléments "simpleType" et "complexType" et en les définissant en dehors de toute définition d'élément ou d'attribut.
- Possibilité de dériver un type à partir d'un autre
  - Restriction/extension
- Exemple
  - Un type de données appelé "nameType" et qui sera une chaîne de caractères acceptant un maximum de 32 caractères

```
<xsd:simpleType name="nameType">
  <xsd:restriction base="xsd:string">
    <xsd:maxLength value="32"/>
  </xsd:restriction>
</xsd:simpleType>
```
  - Un type ISBN sous la forme d'une chaîne de dix caractères numériques

```
<xsd:simpleType name="isbnType">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[0-9]{10}"/>
  </xsd:restriction>
</xsd:simpleType>
```

## 3eme style de schéma: exemple #1

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <!-- definition of simple types -->
  <xsd:simpleType name="nameType">
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="32"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="sinceType">
    <xsd:restriction base="xsd:date"/>
  </xsd:simpleType>
  <xsd:simpleType name="descType">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>
  <xsd:simpleType name="isbnType">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[0-9]{10}" />
    </xsd:restriction>
  </xsd:simpleType>
  - - -
```

## 3eme style de schéma: exemple #2

```
---  
<!-- definition of complex types -->  
<xsd:complexType name="characterType">  
  <xsd:sequence>  
    <xsd:element name="name" type="nameType"/>  
    <xsd:element name="friend-of" type="nameType"  
      minOccurs="0" maxOccurs="unbounded"/>  
    <xsd:element name="since" type="sinceType"/>  
    <xsd:element name="qualification" type="descType"/>  
  </xsd:sequence>  
</xsd:complexType>
```

## 3eme style de schéma: exemple #3

```
---  
<xsd:complexType name="bookType">  
  <xsd:sequence>  
    <xsd:element name="title" type="nameType"/>  
    <xsd:element name="author" type="nameType"/>  
    <!-- the definition of the "character" element is using the  
    "characterType" complex type -->  
    <xsd:element name="character" type="characterType"  
      minOccurs="0" maxOccurs="unbounded"/>  
  </xsd:sequence>  
  <xsd:attribute name="isbn" type="isbnType" use="required"/>  
</xsd:complexType>  
<!-- Reference to "bookType" to define the "book" element -->  
<xsd:element name="book" type="bookType"/>  
</xsd:schema>
```



# Groupes d'éléments (ou d'attributs) #1

- Il est possible de créer des groupes d'éléments et d'attributs et de les utiliser pour définir des types complexes

- Exemple de définition d'un groupe d'éléments

```
<xsd:group name="mainBookElements">  
  <xsd:sequence>  
    <xsd:element name="title" type="nameType"/>  
    <xsd:element name="author" type="nameType"/>  
  </xsd:sequence>  
</xsd:group>
```

- Exemple définition d'un groupe d'attributs

```
<xsd:attributeGroup name="bookAttributes">  
  <xsd:attribute name="isbn" type="isbnType" use="required"/>  
  <xsd:attribute name="available" type="xsd:string"/>  
</xsd:attributeGroup>
```

## Groupes d'éléments (ou d'attributs) #2

- Exemple d'utilisation de groupes d'éléments et d'attributs dans la définition de types complexes

```
<xsd:complexType name="bookType">
  <xsd:sequence>
    <xsd:group ref="mainBookElements"/>
    <xsd:element name="character" type="characterType"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attributeGroup ref="bookAttributes"/>
</xsd:complexType>
```

# Dérivation par **union**

- La définition suivante permet au type de données ISBN d'accepter également les valeurs TBD et NA

```
<xsd:simpleType name="isbnType">  
  <xsd:union>  
    <xsd:simpleType>  
      <xsd:restriction base="xsd:string">  
        <xsd:pattern value="[0-9]{10}" />  
      </xsd:restriction>  
    </xsd:simpleType>  
    <xsd:simpleType>  
      <xsd:restriction base="xsd:NMTOKEN">  
        <xsd:enumeration value="TBD" />  
        <xsd:enumeration value="NA" />  
      </xsd:restriction>  
    </xsd:simpleType>  
  </xsd:union>  
</xsd:simpleType>
```

# Dérivation par **list**

- Le type de données isbnType peut être à son tour utilisé pour constituer un nouveau type (isbnTypes) qui sera une liste séparée par des espaces de valeurs de type "isbnType »

```
<xsd:simpleType name="isbnTypes">  
  <xsd:list itemType="isbnType"/>  
</xsd:simpleType>
```

- Une dérivation par restriction peut ensuite être appliquée pour restreindre le nombre des valeurs dans la liste qui devra être compris entre 1 et 10

```
<xsd:simpleType name="isbnTypes10">  
  <xsd:restriction base="isbnTypes">  
    <xsd:minLength value="1"/>  
    <xsd:maxLength value="10"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

# Types de contenu / contenu vide

- Le type de contenu déjà vu permet de définir des documents XML de type "données"
  - Les types complexes ont uniquement des noeuds attributs et éléments et les types simples des noeuds texte.
- W3C XML Schema supporte également la définition d'autres modèles de contenus
  - Vide, simple avec attributs et mixtes.
- Les éléments à contenu vide (c'est à dire ne pouvant contenir que des attributs) sont définis simplement en omettant de définir des éléments dans un type complexe.
  - La construction suivante définit ainsi un élément "book" à contenu vide acceptant un attribut "isbn »

```
<xsd:element name="book">  
  <xsd:complexType>  
    <xsd:attribute name="isbn" type="isbnType"/>  
  </xsd:complexType>  
</xsd:element>
```

# Type de contenu simple

- Éléments à contenu simple
  - Ne contenant que du texte et des attributs
  - Sont dérivés des types de données simples en utilisant l'élément **<simpleContent>**
- L'élément book peut être modifié de la manière suivante pour accepter également un contenu de type chaîne de caractères

```
<xsd:element name="book">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="isbn" type="isbnType"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```
- La position de la définition de l'attribut indique que l'extension est effectuée par l'ajout de l'attribut.
- La définition précédente acceptera, par exemple, l'élément XML suivant

```
<book isbn="0836217462">
  Funny book by Charles M. Schulz. Its title (Being a Dog Is a Full-Time Job) says it all !
</book>
```

# Type de contenu mixte

- Les modèles de contenu mixtes sont supportés au moyen de l'attribut "mixed" qui doit être placé dans l'élément "complexType"

```
<xsd:element name="book">
  <xsd:complexType mixed="true">
    <xsd:all>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="author" type="xsd:string"/>
    </xsd:all>
    <xsd:attribute name="isbn" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

- Le modèle précédent décrit un élément XML de la forme suivante

```
<book isbn="0836217462">
  Funny book by <author>Charles M. Schulz</author>. Its title
  (<title>Being a Dog Is a Full-Time Job</title>) says it all !
</book>
```

# Contraintes référencielles: unicité #1

- W3C XML Schema plusieurs fonctionnalités s'appuyant sur XPath et permettant de décrire des contraintes d'unicité et des contrôles référentiels
  - unique, key et keyref
- La déclaration d'unicité utilise l'élément "unique". La déclaration suivante, faite à l'intérieur de l'élément "book" indique que le nom d'un personnage doit être unique

```
<xsd:unique name="charName">  
  <xsd:selector xpath="character"/>  
  <xsd:field xpath="name"/>  
</xsd:unique>
```
- La position de la déclaration de cette contrainte (à l'intérieur de la définition de l'élément "book") donne le contexte à partir duquel le contrôle sera effectué (l'élément "book").



## Contraintes référencielles: unicité #2

- Le choix du contexte du contrôle détermine également la portée du test qui sera effectué
  - Dans l'exemple précédent, le nom du personnage sera unique à l'intérieur d'un élément "book"
    - Mais pourra être répété dans un autre élément "book".
- Les deux chemins XPath spécifiés dans l'élément "unique" seront évalués par rapport à ce nœud contexte.
  - Le 1<sup>er</sup> d'entre eux est défini dans l'élément "selector"
    - Il sélectionne l'élément qui doit être unique et doit donc pointer un nœud de type élément.
  - Le 2eme chemin XPath est évalué par rapport au nœud sélectionné
    - Il spécifie le nœud identifiant l'élément qui doit être unique.
    - Il peut pointer un nœud de type élément ou attribut.
    - C'est ce nœud dont l'unicité va être testée.

## Contraintes référencielles: “clé primaire”

- La déclaration "key" est similaire à "unique" et spécifie en outre que cette valeur unique pourra être utilisée comme une clé primaire

- Ce qui lui donne 2 contraintes supplémentaires.

- Pour spécifier que le nom d'un personnage est une clé primaire, il suffit de remplacer "unique" par "key"

```
<xsd:key name="charName">  
  <xsd:selector xpath="character"/>  
  <xsd:field xpath="name"/>  
</xsd:key>
```

# Contraintes référencielles: "référence à une clé primaire"

- La déclaration "keyref" définit une référence à une clé.
- On pourra ainsi tester que le nom inclus dans l'élément "friend-of" correspond à un personnage du même livre

```
<character>
```

```
  <name>Snoopy</name>
```

```
  <friend-of>Peppermint Patty</friend-of>
```

```
  <since>1950-10-04</since>
```

```
  <qualification> extroverted beagle </qualification>
```

```
</character>
```

- Pour cela, on ajoutera un élément "keyref" sous la définition de l'élément "book", au même niveau que l'élément "key"

```
<xsd:keyref name="charNameRef" refer="charName">
```

```
  <xsd:selector xpath="character"/>
```

```
  <xsd:field xpath="friend-of"/>
```

```
</xsd:keyref>
```

# Documentation de schémas

- Il est possible de documenter un schéma
- 2 types de documentations
  - Celle à l'intention des humains peut être définie dans des éléments **<documentation>**
  - Celle à l'intention de programmes doivent être incluses dans des éléments **<appinfo>**
- Les 2 éléments précédents doivent être placés dans un élément **<annotation>**
- Ils acceptent des attributs optionnels "xml:lang" et "source" et tout modèle de contenu.
  - L'attribut "source" est une référence à une URI qui peut être utilisée pour identifier l'objectif du commentaire ou de l'information.
- Exemple
  - Voir le slide suivant

# Documentation de schémas: exemple

```
<xsd:element name="book">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      Top level element
    </xsd:documentation>
    <xsd:documentation xml:lang="fr">
      Element racine.
    </xsd:documentation>
    <xsd:appInfo source="http://example.com/foo/">
      <bind xmlns="http://example.com/bar/">
        <class name="Book"/>
      </bind>
    </xsd:appInfo>
  </xsd:annotation>
```

---

# Réutilisation de définitions

- W3C XML Schema a prévu deux mécanismes d'inclusion de schéma
  - Eléments `<include>` et `<redefine>`
- `<include>` est similaire à un copié/collé des définitions contenues dans le schéma qui est inclus.
  - Les définitions ne peuvent donc pas être redéfinies dans le schéma effectuant l'inclusion. Un exemple:
    - `<xsd:include schemaLocation="character.xsd"/>`
- `<redefine>` est semblable à `<include>`
  - Mais permet de redéfinir des déclarations effectuées dans le schéma inclus. Un exemple:

```
<xsd:redefine schemaLocation="character12.xsd">
  <xsd:simpleType name="nameType">
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="40"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:redefine>
```
  - Les déclarations qui sont redéfinies doivent l'être à l'intérieur de l'élément `<redefine>` .

# Réutilisation de définitions: éléments abstraits et groupes de substitution

- Soit la définition suivante

```
<xsd:element name="name" type="xsd:string"/>  
<xsd:element name="surname" type="xsd:string"  
  substitutionGroup="name" />
```

- Crée un groupe de substitutions formé des éléments "surname" et "name" et ces éléments peuvent être utilisés indifféremment partout où "name" a été défini.

- La définition suivante

```
<xsd:element name="nameelt" type="xsd:string" abstract="true"/>  
<xsd:element name="name" type="xsd:string"  
  substitutionGroup="nameelt"/>  
<xsd:element name="surname" type="xsd:string"  
  substitutionGroup="nameelt"/>
```

- Définit "nameelt" comme un élément abstrait qui doit être remplacé par "name" ou "surname" lorsqu'il est employé dans un document.

# Réutilisation de définitions: éléments "finaux"

- W3C XML Schema permet également de définir des éléments et des types complexes comme étant "finaux" en utilisant l'attribut **final**.
  - Cet attribut peut prendre les valeurs "restriction", "extension" ou "#all"
    - Bloque, respectivement, les dérivations par restriction, extension ou toutes les dérivations.
  - Exemple: Interdiction de toute dérivation du type "characterType".
    - `<xsd:complexType name="characterType" final="#all">`
- L'attribut "final" ne s'applique qu'aux éléments et aux types complexes
  - Un mécanisme encore plus fin permet de contrôler la dérivation de types simples facette par facette.
    - L'attribut correspondant ("fixed") s'applique aux différentes facettes et lorsque sa valeur est "true", la facette ne peut plus être restreinte.
  - L'exemple suivant montre comment on peut interdire toute restriction ultérieure sur la taille maximum du type "nameType"
 

```
<xsd:simpleType name="nameType">
  <xsd:restriction base="xsd:string">
    <xsd:maxLength value="32" fixed="true"/>
  </xsd:restriction>
</xsd:simpleType>
```



# Espaces de noms #1

- Regroupement logique d'un ensemble d'éléments et de types
- Leur support est simple et souple et est la principale motivation du développement de W3C XML Schema
- Chaque schema peut être lié à un espace de nom
- Il est possible de définir des éléments et attributs sans espace de nom dans un schema

# Espaces de noms #2

## ■ Déclaration dans l'élément <schema>

```
<xsd:schema
```

```
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema" <!--définition du  
préfixe xsd-->
```

```
xmlns="http://example.org/ns/books/" <!--Espace de nom par défaut-->
```

```
targetNamespace="http://example.org/ns/books/" <!--Espace de  
nom cible-->
```

```
elementFormDefault="qualified" <!--Valeur par défaut de l'attribut form des  
éléments-->
```

```
attributeFormDefault="unqualified"> <!--Valeur par défaut de l'attribut form  
des attributs-->
```

## ■ Un élément book dans un document XML

```
□ <book                                     isbn="0836217462"  
    xmlns="http://example.org/ns/books/">
```

# Espaces de noms #3

- Comment utiliser des objets appartenant à d'autres espaces de noms que les deux vus précédemment (XML Schema et le namespace par défaut).
  - Exemple: On veut utiliser des éléments de l'espace de noms XML 1.0
- 3 étapes
  - 1ere étape: définir un préfixe pour cet espace de noms de manière standard, par exemple

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
targetNamespace="http://example.org/ns/books/"
xmlns:xml="http://www.w3.org/XML/1998/namespace"
elementFormDefault="qualified" >
```
  - 2eme étape: indiquer où trouver le schéma correspondant à cet espace de nom en utilisant l'élément "import" :

```
<import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="myxml.xsd"/>
```
  - 3eme étape: utilisation d'objet préfixé par « xml »

```
<xsd:element name="title">
<xsd:complexType> <xsd:simpleContent> <xsd:extension base="xsd:string">
<xsd:attribute ref="xml:lang"/>
</xsd:extension> </xsd:simpleContent> </xsd:complexType>
</xsd:element>
```

# Espaces de noms #4

## ■ Eléments <any> et <anyAttribute>

- Permettent d'autoriser des éléments ou attributs non déterminés appartenant à d'autres espaces de noms.
- Exemple: si on veut que le type descType accepte n'importe quel élément XHTML:

```
<xsd:complexType name="descType" mixed="true">
```

```
  <xsd:sequence>
```

```
    <xsd:any namespace=http://www.w3.org/1999/xhtml  
      minOccurs="0" maxOccurs="unbounded" processContents="skip"/>
```

```
  </xsd:sequence>
```

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

- descType devient ainsi un type à contenu mixte acceptant un nombre quelconque d'éléments de l'espace de noms "<http://www.w3.org/1999/xhtml> namespace".

# Espaces de noms #5

- L'attribut **processContents** peut prendre plusieurs valeurs
  - "skip"
    - Indique qu'on ne souhaite pas valider les éléments par rapport à un schéma pour XHTML
  - "strict"
    - Impose une validation
  - "lax"
    - Demande de valider si le schéma correspondant est disponible
- L'attribut **namespace** accepte les valeurs suivantes
  - Un URI ou une liste d'URIs séparées par des espaces
  - "##any"
    - N'importe quel espace de noms
  - "##local"
    - Éléments non qualifiés
  - "##targetNamespace"
    - L'espace de noms cible
  - "##other"
    - Tout espace de noms autre que la cible

# Les schémas dans les documents XML

- Des extensions sont utilisées dans les documents XML pour supporter les XML schema.
  - Elles sont identifiées par l'espace de noms spécifique <http://www.w3.org/2000/10/XMLSchemainstance>
    - Souvent associé au préfixe "xsi »
- Deux attributs de cet espace de noms permettent de lier un document à des schémas W3C XML Schema
  - **noNamespaceSchemaLocation**
    - Schéma ne définissant pas d'espace de noms
    - Exemple

```
<book isbn="0836217462" xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="file:library.xsd">
```
  - **schemaLocation**
    - Schéma définissant un espace de noms
    - Exemple

```
<book isbn="0836217462" xmlns="http://example.org/ns/books/"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:schemaLocation="http://example.org/ns/books/ file:library.xsd">
```

## Interrogation n° 2

1. Donner la forme générale de définition d'un type complexe en XML Schema
2. Dire (un exemple à l'appui) la différence entre l'héritage par extension et l'héritage par restriction en XML Schema
3. A travers un exemple définir la notion de groupe de substitution de XML Schema

# Etude de cas

Penser à concevoir et réaliser une application de gestion de ressources XML

- ☐ Validation, mise en forme, exploitation, etc.

On prendra pour exemples les problèmes suivants

- ☐ La gestion des chaînes de restaurants
- ☐ La gestion des examens

Seance 1 (15 min)

- ☐ Spécifier le besoin





Chapitre 4

# Mise en forme et transformation de documents XML

DTDs, W3 XML Schemas.

# Les feuilles de styles CSS

- Apparues vers les années 1990s
- Standard W3C qui s'est imposé au détriment de la "confrontation" Netscape Navigator/Internet Explorer
  - Chacun créait ses propres balises de mise en forme
- Plusieurs versions
  - CSS1, CSS2, CSS3
- Leur champ d'action ne s'étend pas uniquement au médium screen
  - Autres médias: print, projection, braille, aural/speech, ...
- Traitement en cascade ! [CSS stands for «*Cascading Style Sheet*»]
  - Toutes les balises descendantes héritent des propriétés de mise en forme de leurs balises mères

Bienvenue dans le XXI<sup>ème</sup> siècle !

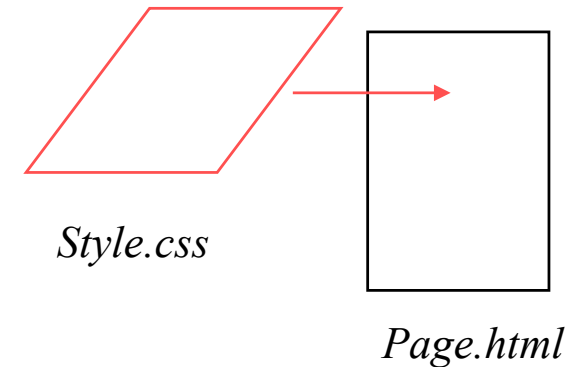
L'eau est notée H<sub>2</sub>O en chimie.

# Rôle des feuilles de styles

- Mise en forme des pages Web

- Séparation du fond et de la forme

- plusieurs vertues



Bienvenue dans le XXI<sup>ème</sup> siècle !

L'eau est notée H<sub>2</sub>O en chimie.

- Contribuer à la rapidité de la navigation sur le Web

- Les feuilles de styles sont conservées dans le cache du client

- Leur utilisation est imposée en XHTML

# Localisation des feuilles de styles

## ■ Balise STYLE

- En entête de page
- `<style type="text/css">....</style>`

## ■ Balise LINK

- En entête de page
- `<link rel="stylesheet" type="text/css" href="url"/>`

Bienvenue dans le XXI<sup>ème</sup> siècle !

L'eau est notée H<sub>2</sub>O en chimie.

## ■ Attribut STYLE des éléments HTML

- Dans tout le corps du document
- `<baliseOuvranteOuVide style="...">`

# Syntaxe des feuilles de styles

- Différente de HTML
- Une feuille de styles est un ensemble de règles
  - **Forme d'une règle**
    - Un sélecteur
      - Élément ou groupe d'éléments HTML
    - Une déclaration
      - Liste d'associations attributs:valeurs séparées par des “;”, le tout entre accolades
  - **Exemple de règle**
    - `h1 { color: blue; text-align: center; }`
- Non sensibles à la casse
  - Il est conseillé de les écrire en lettres minuscules

Bienvenue dans le XXI<sup>ème</sup> siècle !  
L'eau est notée H<sub>2</sub>O en chimie.

# La règle @import

- Intégrée depuis CSS2

```
<style type="text/css">  
@import url(styles.css);  
</style>
```

Bienvenue dans le XXI<sup>ème</sup> siècle !

L'eau est notée H<sub>2</sub>O en chimie.

- Permet d'inclure des feuilles de styles dans d'autres

- ☐ Ce qui permet de créer des feuilles de styles dynamiques sans devoir recopier plusieurs fois le même code



Différence avec <link>?

# Les différents sélecteurs

- Les sélecteurs de balises
  - Utilisés dans le précédent exemple
- Les sélecteurs de classes
  - Une classe est un nom donné à un ensemble d'éléments HTML à distinguer
- Les sélecteurs d'identifiants
  - Un identifiant ou id est le nom attribué à un élément unique dans le document HTML
- Les pseudo-classes et les pseudo-éléments
  - Variantes pour certaines fonctionnalités, par exemple les liens

Bienvenue dans le XXI<sup>ème</sup> siècle !  
L'eau est notée H<sub>2</sub>O en chimie.

# Exemples CSS #1

## ■ Style interne en HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
<STYLE type="text/css">
    Bienvenue dans le XXIème siècle !
    C'est le siècle de l'Internet et de la chimie.
<!-- Règles sous la forme, par exemple,
    balise {propriétés} -->
</STYLE>
</HEAD>
<BODY>
<balise> ... </balise>

...
</BODY>
</HTML>
```



# Exemples CSS #2

## ■ Style externe en HTML

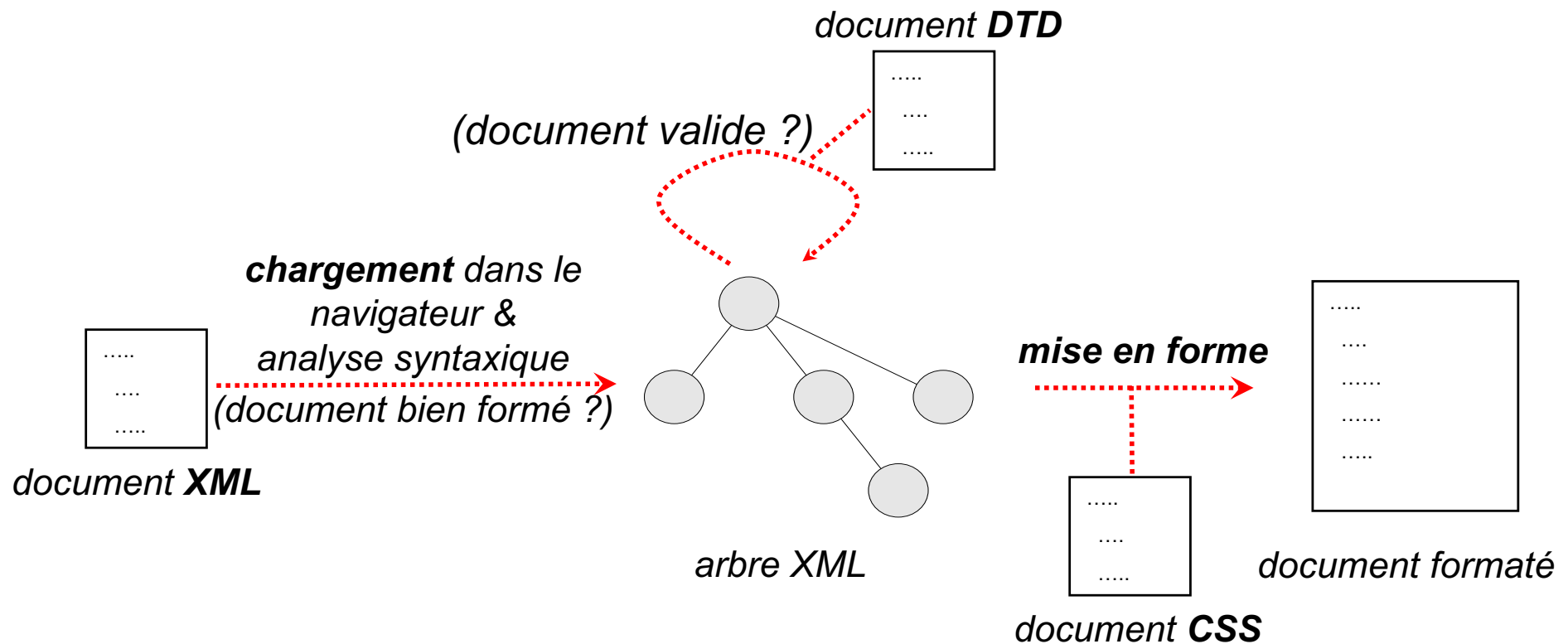
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
body {backgroundimage; home.gif;}
LI {font: 13px Verdana;}
B {font: 14px Verdana; font-weight:bold;}
A { font:12px Verdana; font-weight:bold; color=black; text-decoration:none; }
H1 {font: 16px Arial;font-weight:
    Bienvenue dans le XXIème siècle !
    L'eau est notée H2O en chimie.
    bold;color=black;}
H2 {font: 14px Arial;font-weight:bold;color=black;}
```

## □ Style externe en HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
<LINK rel="stylesheet" type="text/css" href="FichierDeStyles.css">
....
</HEAD>
...
```

# Présentation des documents XML avec CSS

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





# Définition des styles CSS

- Tous les titres de niveau 1 et 2 **<H1>** et **<H2>**

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

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

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

- Tous 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;}
```

# Exemple XML & CSS

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

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

*Fichier : exemple.xml*

```
<?xml-stylesheet href="/exemple.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 >
```

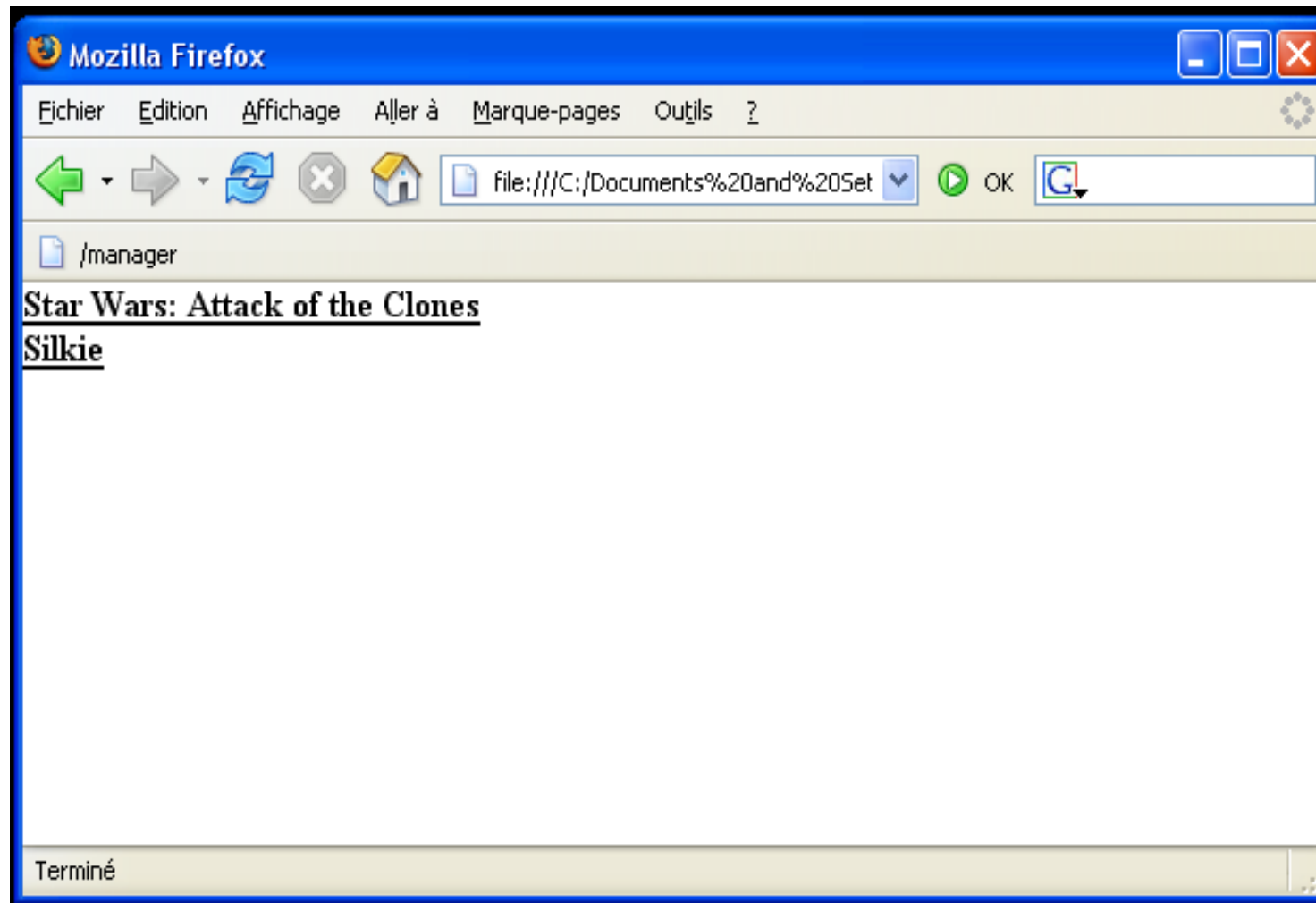
```
livre[categorie="Literature"] { display:none; }
```

```
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;}
```

*Fichier : exemple.css*

# Présentation dans un navigateur



# Exercice à rendre

- Reprendre l'exercice sur le site portail qui propose aux restaurants un espace de diffusion
  - Construire une DTD et un schema pour les documents XML qui permettent de décrire les restaurants souscripteurs
  - Proposer une feuille de style CSS permettant l'affichage des informations sur un restaurant

# Limites de CSS

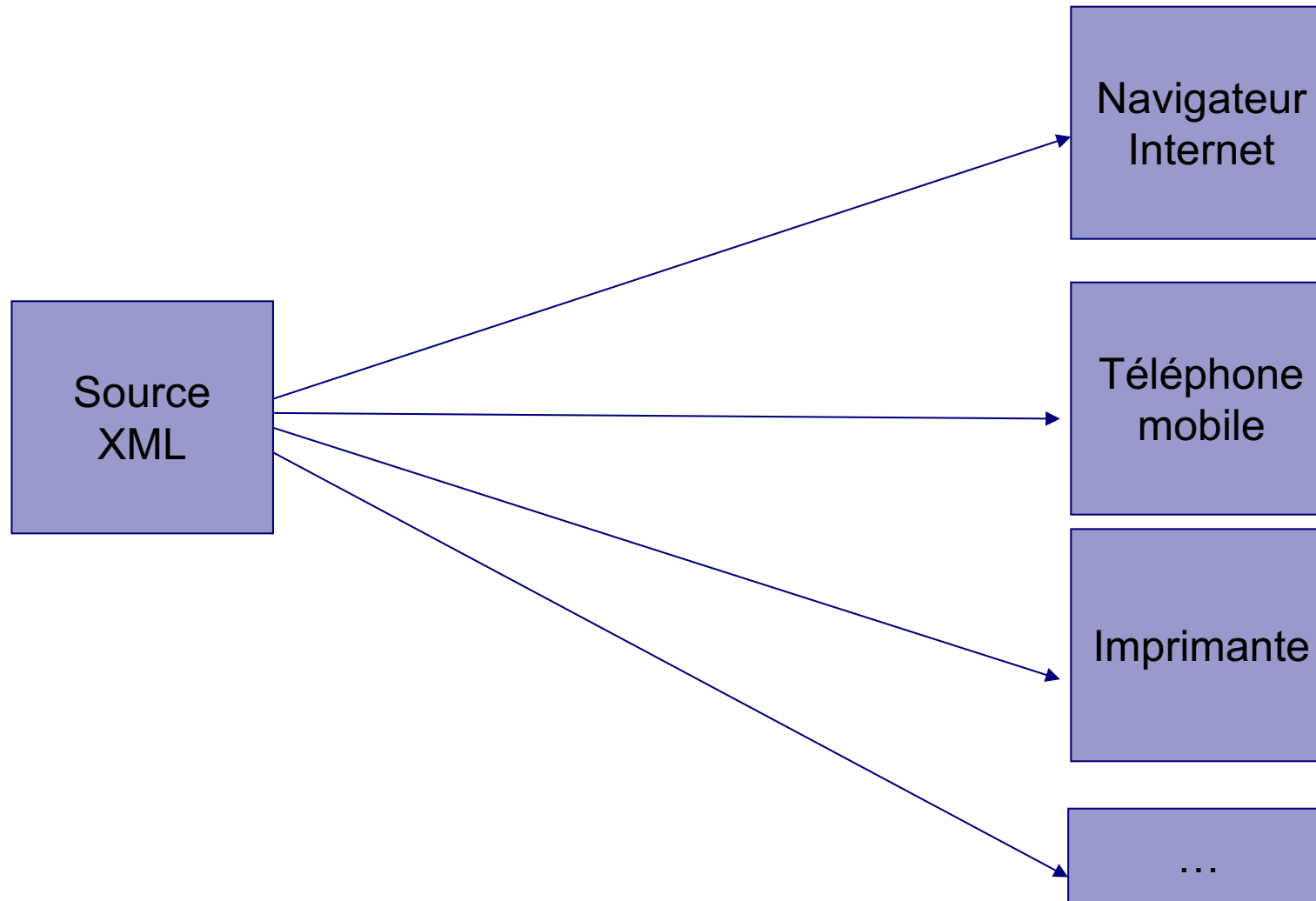
- Elles ne peuvent pas manipuler les éléments XML
  - Pas, par exemple, de sélection ou de tri
  - Le document XML est affiché en entier ou pas
- Elles reposent entièrement sur la capacité du navigateur à les exploiter.
  - Le support des CSS varie largement en fonction des navigateurs
- Elles ne permettent pas une variation de la mise en forme en fonction des attributs des éléments
  - Même si on a des avancés avec les récentes versions
- Elles sont uniquement orientées « affichage de document »

# Transformation de documents XML #1

- XML permet de séparer le contenu et la présentation
- Cette séparation a plusieurs avantages
  - Possibilité d'extraire des données du document
  - Possibilité de transformer le contenu
  - Possibilité de publier le document sous différents formats



# Transformation de documents XML #2



# Transformation de documents XML #2

- XML

- `<titre>Information</titre>`

- HTML

- `<h1>Information</h1>`

- WML

- `<wml><card>Information</card></wml>`

- Version Imprimable

- `<fo:block>Information</fo:block>`

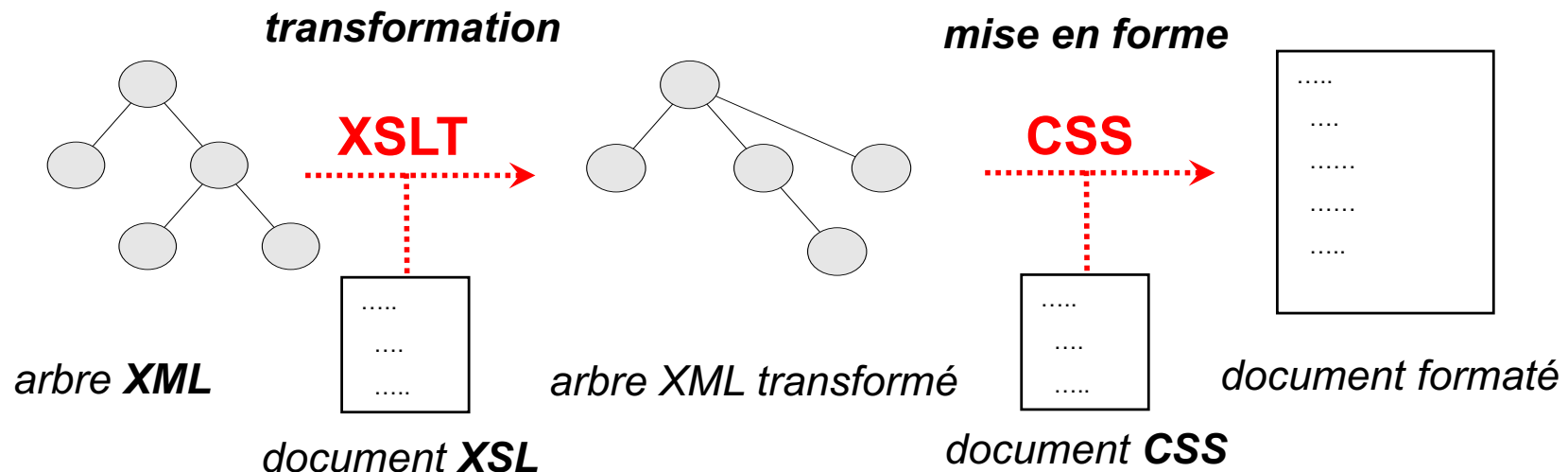
- ...

# Transformation de documents XML #2

- Du fait de l'importance des transformations dans toute application XML, il a été mis en place d'un nouveau langage
  - eXtensible Stylesheet Language Transformations

# eXtensible StyleSheet Language (XSL)

- **XSL = XSLT (+CSS) + XSLFO + Xpath**
  - Plus qu'une simple mise en forme ...
- **CSS** → Mise en forme de fichier XML
- **XSLT** (XSLTransformations) → Transformation de l'arbre XML



# Xpath

- Langage utilisé pour trouver de l'information à partir d'un fichier XML
  - Navigation dans un document XML
- Définit des fonctions standards
- Est un élément majeur de XSLT
  - Comme d'autres standards W3C
- Est une recommandation W3C

# Terminologie Xpath

- Un document comporte des nœuds et des valeurs atomiques
  - Les nœuds
    - elements, attributs, textes, namespaces, instruction de traitements, commentaires, noeuds documents.
  - Les valeurs atomiques
    - Nœuds sans parent ni enfant
- Il existe des relations entre les nœuds d'un document
- Les noeuds permettent à un processeur XSL
  - Navigation dans un l'arborescence d'un document
  - Sélection d'éléments et alors ...
    - ... Extraction de valeurs
  - ...

# Terminologie Xpath: relations entre noeuds

- Parent (parent)
  - Chaque élément ou attribut a un parent
- Enfant (children)
  - Les nœuds éléments peuvent avoir 0, 1 ou + enfants
- Frères (siblings)
  - Nœuds ayant le même parent
- Ancêtre (ancestor)
  - Un parent d'un nœud, le parent de son parent, ...
- Descendant (descendant)
  - Un enfant d'un nœud, l'enfant de son enfant, ...
- ...

**Ces relations permettent de définir des axes nodaux**

## Axes nodaux

- Permettent la sélection, à partir du noeud courant, des parties du document XML
  - Ouvrent des directions de recherche
- Forme générale
  - `préfixe::pattern`
    - où préfixe indique la direction de recherche
- Exemple
  - `ancestor::personne/@age`
    - Désigne l'age (attribut) de l'élément personne ancêtre du noeud courant



# Axes nodaux: les différents préfixes

- **self::**
  - Le noeud courant lui-même
- **ancestor::**
  - Les noeuds parents du noeud courant
- **ancestor-or-self::**
  - Les noeuds parents du noeud courant plus le noeud courant
- **parent::**
  - Les noeuds parents directs du noeud courant
- **descendant::**
  - Les noeuds descendants du noeud courant
- **descendant-or-self::**
  - Les noeuds parents du noeud courant plus le noeud courant

# Axes nodaux: les différents préfixes

- **child::**
  - Les noeuds fils directs du noeud courant
- **attribute::**
  - Les attributs du noeud courant
- **following::**
  - Les noeuds suivants (après la fermeture) du noeud courant
- **following-sibling::**
  - Les noeuds suivants du noeud courant, de même parent
- **preceding::**
  - Les noeuds précédants (avant le “début”) du noeud courant
- **preceding-sibling::**
  - Les noeuds précédents du noeud courant, de même parent

# Axes nodaux abrégés #1

- **element**
  - Tous les éléments element fils du nœud courant (child::element).
- **\***
  - Tous les éléments fils du noeud courant
- **/**
  - L'élément racine
- **//**
  - N'importe quel descendant de l'élément racine, donc tous les éléments (descendant-or-self::node()).
- **.**
  - L'élément courant (self::node()).
- **..**
  - Permet de remonter d'un niveau dans l'arborescence du document par rapport à l'élément courant (parent::node()).

# Axes nodaux abrégés #2

- `/element`
  - Tous les éléments `element` sous l'élément racine.
- `./element`
  - Tous les éléments `element` sous l'élément courant
- `../element`
  - Tous les éléments `element` sous l'élément parent du noeud courant.
- `//element`
  - Tous les éléments `element` descendants du noeud courant à quelque niveau de profondeur que ce soit.
- `@attribut`
  - Tous les attributs `attribut` du noeud courant (`attribute::attribut`).
- `|`
  - Correspond à un ou.

# Fonctions nodales

- `position()`
  - Retourne la position du noeud courant à l'intérieur du noeud parent.
- `count(ensemble_noeud)`
  - Retourne le nombre de noeuds dans l'ensemble de noeuds passé en argument.
- `current()`
  - Retourne le noeud courant.
- .... [Xpath définit plusieurs fonctions]

## Exemple

<h3>

Nombre de personnes : <xsl:value-of select=" count(//personne) "/>

</h3>

# Opérateurs

## ■ Booléens

- or, and, not()

  - Logiques

- |

  - Sélection de plusieurs motifs.

- =, !=, <, <=, >, >=

  - Comparaison

## ■ Calculs

- +, -, div, mod

# Prédicats

- `element`
  - Tous les éléments `element` fils du noeud courant.
- `element[n]`
  - Le nième élément `element` dans le noeud courant.
- `element[elt]`
  - Dans le noeud courant, l'élément `element` qui a comme élément fils `elt`.
- `[elt="valeur"]`
  - Dans le noeud courant, l'élément ayant pour fils un noeud `elt` qui a une valeur égale à `valeur`.
- `element[@attribut]`
  - Dans le noeud courant, l'élément `element` qui possède un attribut `attribut`.
- `[@attribut='valeur']`
  - Dans le noeud courant, l'élément dont l'attribut `attribut` a une valeur égale à `valeur`.

# XSLT

1. **`/descendant::figure[position()=42]`**
2. **`/child::doc/child::chapter[position()=5]/child::section[position()=2]`**
3. **`child::chapter/descendant::para`**
4. **`child::* / child::para`**
5. **`attribute::*`**

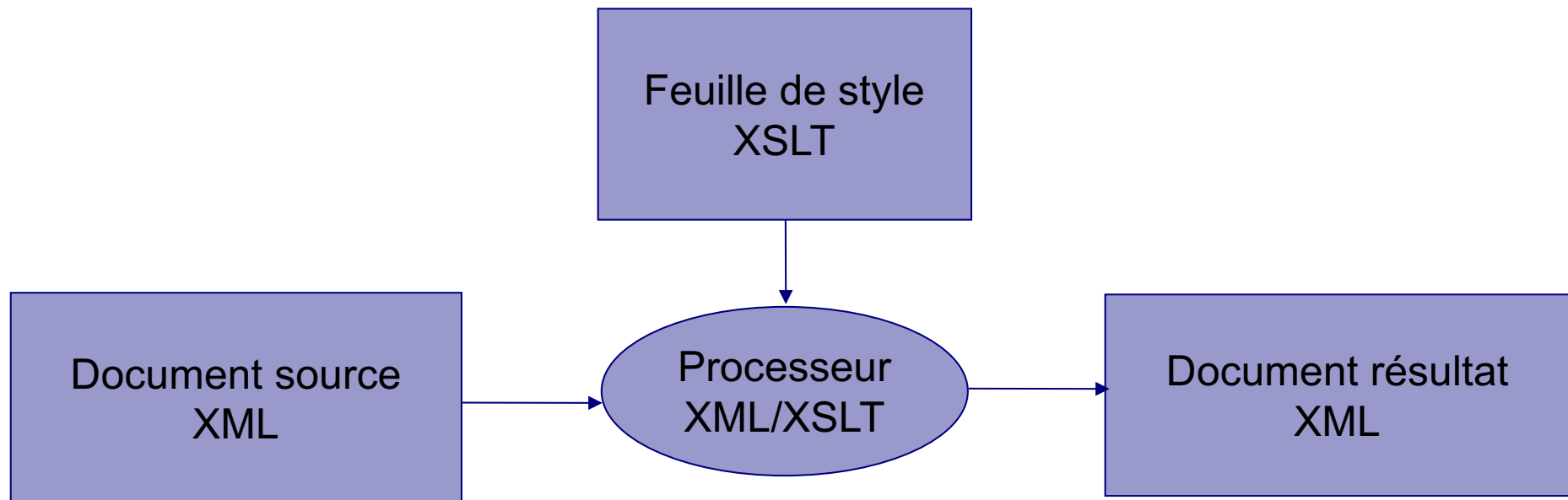


# XSLT

## ■ XSLT permet

- ☐ La transformation des documents XML d'un vocabulaire dans un autre
- ☐ La sélection de contenu spécifique dans un document XML
- ☐ Le tri du contenu d'un élément XML
- ☐ La combinaison du contenu de différents documents XML
- ☐ ...

# Diagramme de transformation d'un document XML



# Principes XSLT

- Une feuille de style XSL est un document XML valide
- Le même éditeur que pour le document XML peut être utilisé
  - Vérifier que la feuille de style est bien formé
  - Pas de DTD pour les feuilles de styles!
- XSLT est un langage déclaratif
  - Possède des fonctionnalités très puissantes
  - Différent des langages procéduraux
    - Déclaratif
      - Un ensemble de templates ou modeles
        - Composé d'éléments et d'attributs dans l'arbre source
      - Définit l'arbre résultant

# Principes XSLT

- Une feuille de style XSL est un document XML valide
- Le même éditeur que pour le document XML peut être utilisé
  - Vérifier que la feuille de style est bien formé
  - Pas de DTD pour les feuilles de styles!
- XSLT est un langage déclaratif
  - Possède des fonctionnalités très puissantes
  - Différent des langages procéduraux
    - Déclaratif
      - Un ensemble de templates ou modèles
        - Composé d'éléments et d'attributs dans l'arbre source
      - Définit l'arbre résultant

# Exemple XSLT

- Fragment XML

```
<message>Ceci est un message important !</message>
```

- Fragment XSLT

```
<xsl:template match="message">  
  <H1><xsl:value-of select="." /></H1>  
</xsl:template>
```

- Fragment résultant

```
<H1>Ceci est un message important !</H1>
```

# Notre première feuille de style XSLT #1

- Spécifier le vocabulaire d'une feuille de style
  - Doit posséder un élément racine

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

ou

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

- Permet d'informer les parseurs de la présence de feuille style XSL
- Que tous les éléments préfixés de xsl: seront des instructions XSL
- Et qu'ils sont conformes à la version 1.0

# Notre première feuille de style XSLT #2

## ■ Exemple de fichier XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<statistiques_equipe>
  <equipe>Chicago Bulls</equipe>
  <victoires>78</victoires>
  <défaites>2</défaites>
  <classement>1</classement>
</statistiques_equipe>
```

# Notre première feuille de style XSLT #3

- Algorithme de la feuille de style
  - ☐ Créer un élément <html>
  - ☐ Créer le titre pour la fenêtre du navigateur comportant le nom de l'équipe
  - ☐ Créer un titre pour la page web, comportant également le nom de l'équipe
  - ☐ Afficher le nom de l'équipe et les résultats



# Notre première feuille de style XSLT #4

## ■ La feuille de style XSLT (*statistiques.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="statistiques_equipe">
<html>
<head>
<title>Statistique des <xsl:value-of select="équipe"/></title>
</head>
<body>
<h1>statistique des <xsl:value-of select="équipe" /></h1>
<p>
les <xsl:value-of select="équipe" />
sont en position <xsl:value-of select="classement"/>
</p>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

## Notre première feuille de style XSLT #5

- Attacher la feuille de style au document XML

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

# Modèle générique de feuille de style XSLT #1

- La plupart des feuilles de styles est constituée de modèles (règles) semblables à celui-ci

```
<xsl:template match="/">  
    [élément littéraux résultants]  
    <xsl:apply-templates />  
    [élément littéraux résultants]  
</xsl:template>
```

# Modèle générique de feuille de style XSLT #2

- Ce modèle réalise les opérations suivantes
  - Trouve l'élément racine du document /
    - / est un raccourci pour désigner l'élément du plus haut niveau hiérarchique
  - Écrit dans le document résultant des éléments littéraux
    - Par exemple du HTML
  - Applique les autres modèles de la feuille de style aux éléments enfants
    - (<xsl:apply-templates/>)
  - Écrit dans le document résultant une deuxième série d'éléments littéraux
  - Et arrête le traitement du document

# Les instructions XSLT

- `<xsl:template>`
- `<xsl:apply-templates>`
- `<xsl:value-of>`
- `<xsl:for-each>`
- `<xsl:choose>`
- `<xsl:if>`
- `<xsl:attribute>`
- `<xsl:sort>`
- `<xsl:copy>`
- `<xsl:comment>`
- ...

# L'instruction `<template>`

- C'est la façon la plus simple de lier une règle de traitement et les éléments à traiter
- Le modèle est construit de la façon suivante
  - Où [modèle] peut être un nom d'élément ou une expression identifiant un fragment du document XML

```
<xsl:template match="[modèle]">  
    [élément résultant littéraux et éléments XSLT]  
</xsl:template>
```

# L'instruction `<apply-templates>`

- S'utilise de deux façons différentes

- Sans attribut

- `<xsl:apply-templates/>`

- Signifie appliquer tous les modèles de cette feuille de style qui s'appliquent à l'élément courant ou à ses fils.

- Avec l'attribut **select**

- `<xsl:apply-templates select="[expression]"/>`

- Signifie appliquer tous les modèles de cette feuille de style qui s'appliquent à l'élément courant ou à ses fils qui vérifient l'expression Xpath spécifiée
        - Souvent utilisé pour définir l'ordre dans lequel les modèles sont exécutés.

**LES EXPRESSIONS XPATH PERMETTENT DE SÉLECTIONNER UN OU PLUSIEURS ÉLÉMENTS D'UN DOCUMENT XML**

## L'instruction `<for-each>`

- Permet d'appliquer un traitement à une série de contenus du document source

- ☐ C'est, par exemple, un moyen simple pour remplir des tableaux HTML avec des contenus d'élément XML

```
<xsl:for-each select="equipe">
```

```
...
```

```
</xsl:for-each>
```



## L'instruction `<value-of>`

- Permet d'écrire dans le document résultant la valeur d'un élément du document source
  - Si l'élément du document source possède des éléments enfants, le texte de ces éléments sont aussi inclus
- Exemple
  - `<xsl:value-of select="victoire"/>`

## L'instruction `<if>`

- Permet d'appliquer un test conditionnel dans la structure d'une feuille de style XSL

- Syntaxe

`<xsl:if test="condition">`

Instructions...

`</xsl:if>`

# L'instruction `<choose>`

- Combiné avec `<xsl:when>` et `<xsl:otherwise>`, permet de construire des tests conditionnels à l'instar du *switch* de Java.
- Syntaxe
  - `<xsl:choose>`
    - `<xsl:when test="condition">`
      - *instructions*
    - ...
    - `</xsl:when>`
    - ...
    - `<xsl:otherwise>`
      - *instructions*
    - ...
    - `</xsl:otherwise>`
  - `</xsl:choose>`

## L'instruction `<attribute>`

- Ajoute à l'élément courant un attribut avec le nom et la valeur indiqués dans l'arborescence d'un document résultant

- Exemple

```
<xsl:attribute name="jour">
```

```
  Lundi
```

```
</xsl:attribute>
```

## L'instruction `<comment>`

- Permet d'insérer un commentaire dans l'arborescence d'un document XML

- Exemple

`<xsl:comment>`

Ce commentaire a été ajouté avec XSLT.

`</xsl:comment>`

## L'instruction `<copy>`

- Est utilisée pour recopier le noeud courant dans l'arborescence du document XML résultant
  - La copie se fait sans les nœuds fils et attributs
- Syntaxe

```
<xsl:copy [use-attribute-sets="liste"]>
  template...
</xsl:copy>
```
- Autre syntaxe

```
<xsl:copy/>
```

## L'instruction `<copy-of>`

- Permet de créer une copie des noeuds sélectionnés par le pattern dans l'arbre du document XML résultant.
  - La copie se fait avec les nœuds fils et attributs
- Syntaxe
  - `<xsl:copy-of select="pattern"/>`

## L'instruction `<variable>`

- Permet de déclarer une « variable » associée à une valeur

- Exemple

```
<xsl:variable name="ligne">
  <tr><td>1</td><td>2</td></tr>
</xsl:variable>

...
<xsl:copy-of select="$ligne" />
```



## L'instruction `<sort>`

- Permet de trier les nœuds sélectionnés
  - S'emploie toujours dans un `<for-each>` ou un `<apply-templates>`

- Exemple

```

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

```

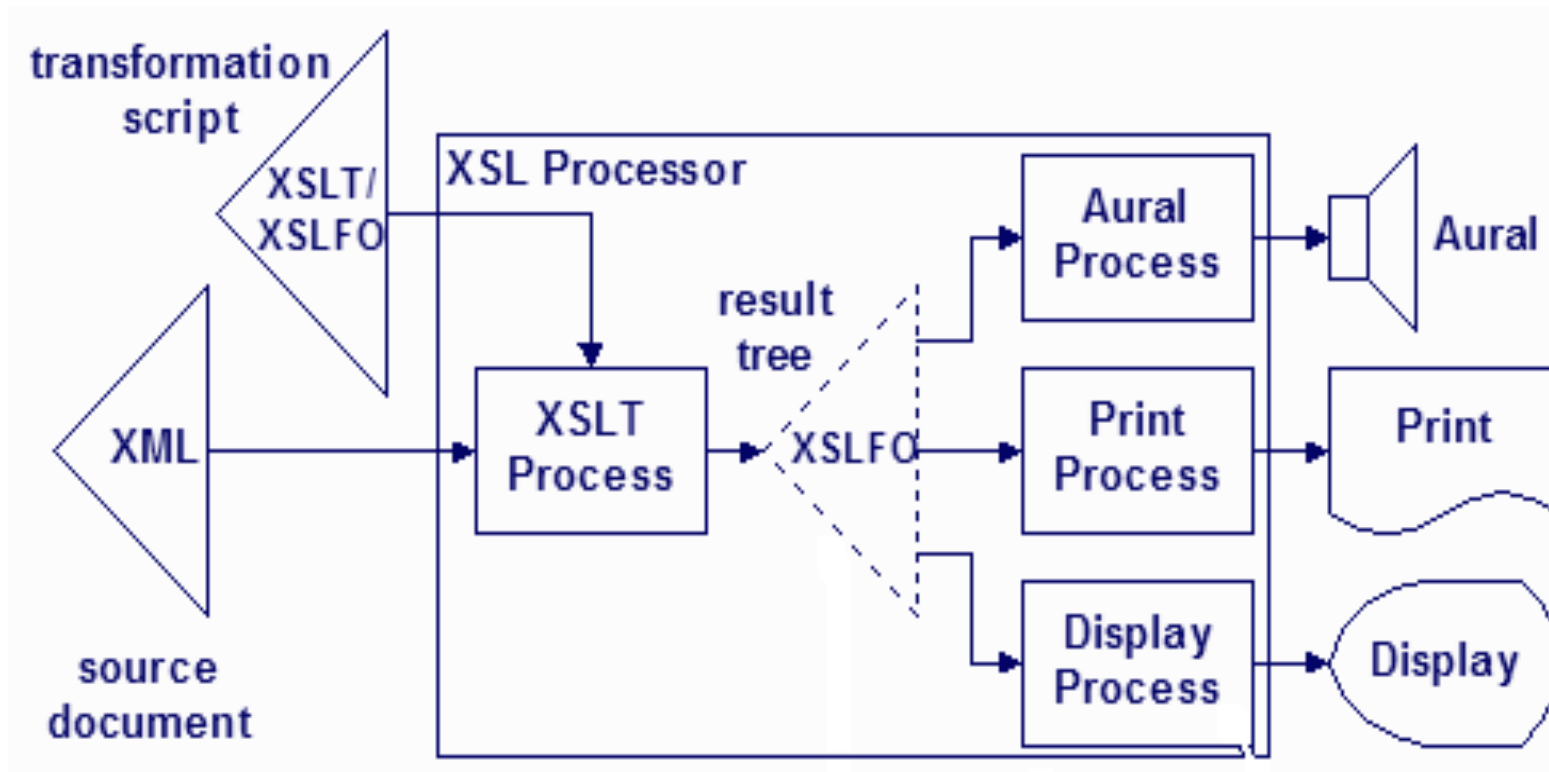
## La fonction `document()`

- L'une des plus importantes des fonctions XSLT
- Utilisée pour accéder à des nœuds externes au document courant
- Exemple
  - `<xsl:value-of  
select="document('menus.xml')/MENU[@JOUR=$jour]"/>`

# XSLFO

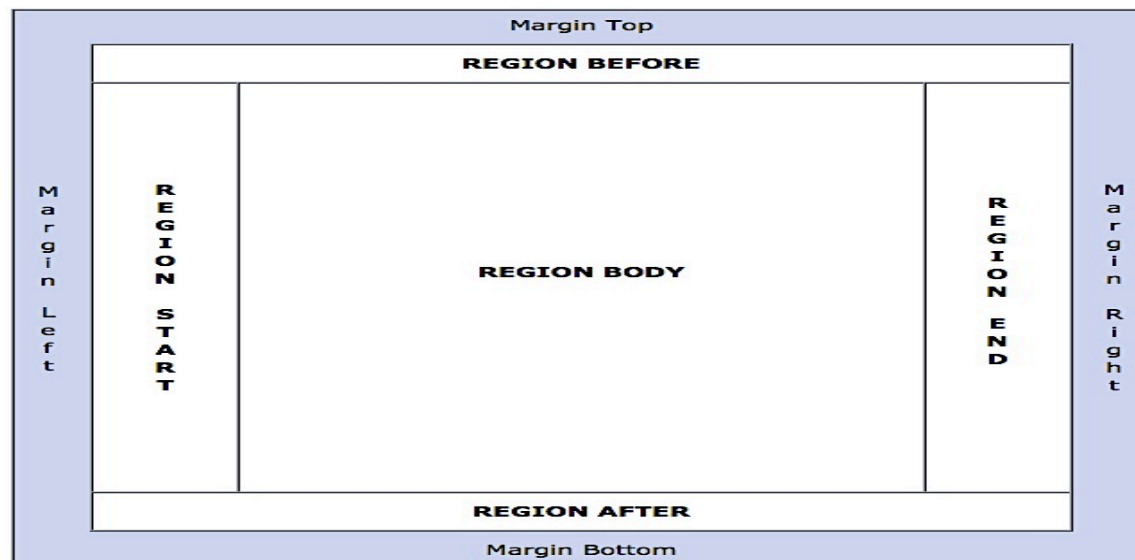
- Langage pour formater du XML pour un écran, du papier ou un autre médium
- XSL-FO est à une qualité d'affichage beaucoup plus élevée que CSS+HTML/XML
  - Multiple colonnes ou layout (plan sur lequel on peut afficher)
  - Formatage conditionnel au niveau des contenus du document
  - Existence de notes de pied de page, de marges, etc.
  - Génération automatique des numéros de page
  - etc.

# XSLFO - Un “big picture”



# XSLFO: le principe

- Les pages
  - Il y'a un modèle général pour les pages
- Les régions d'une page
  - Des margins tout au tour
  - Region-before and Region-after (ie entete et pied de page)
  - Region-start and Region-end (ie les colonnes à coté du cadre central)
  - Region-body (cadre central)



## Structure d'un document XSL-FO

- En premier la description des pages maîtres
  - Cela décrit la structure des pages
- Ensuite la description des séquences de pages
  - Cela fournit le contenu à mettre dans les pages

# Document XSL-FO générique

```
<?xml version="1.0"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-masterset>
    <!--page masters go here -->
  </fo:layout-master-set>
  <fo:page-sequence master-reference="pmname">
    <!--contents to go in pages here -->
  </fo:page-sequence>
</fo:root>
```

# Exemple de document XSL-FO

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="mypage">
      <fo:region-body margin="1in"/>
    </fo:simple-page-master>
  </fo:layout-master-set>
  <fo:page-sequence master-reference="mypage">
    <fo:flow flow-name=" xsl:region-body">
      <fo:block>Hello, world!</fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```



## Autre exemple

- Extrait d'un document XSLFO
  - Page nommée A4 de 297mm/210

```
<fo:simple-page-master master-name="A4" page-width="297mm"
page-height="210mm" margin-top="1cm" margin-bottom="1cm"
margin-left="1cm" margin-right="1cm">
  <fo:region-body margin="3cm"/>
  <fo:region-before extent="2cm"/>
  <fo:region-after extent="2cm"/>
  <fo:region-start extent="2cm"/>
  <fo:region-end extent="2cm"/>
</fo:simple-page-master>
```

# Les objets de formatage: pagination et layout

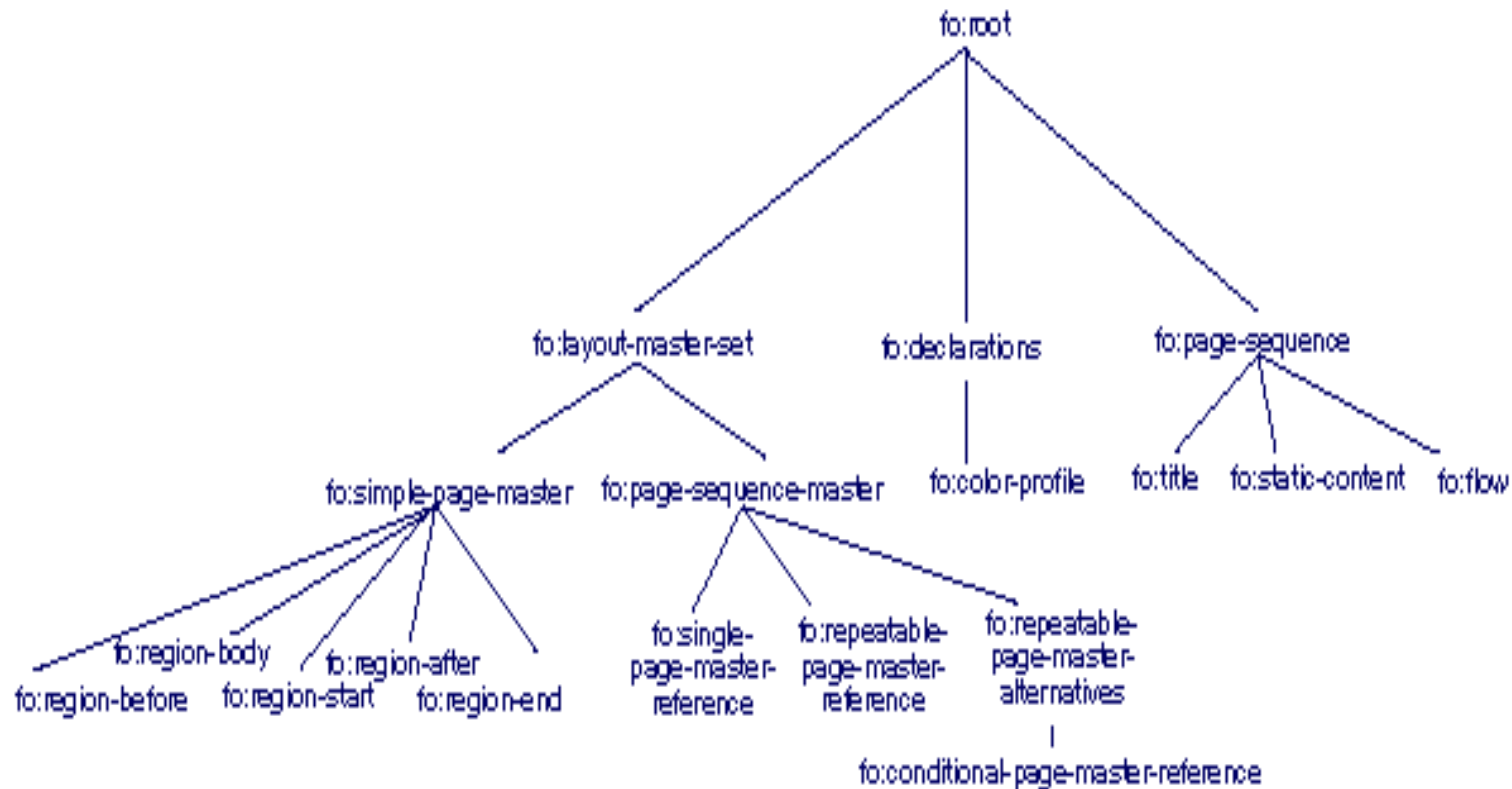
## ■ layout-master-set

- Définit la géométrie et le séquençement des pages
- Peut avoir deux fils
  - **page-master**
    - Géométrie d'une page, sa subdivision.
  - **page-sequence-master**
    - Séquence de page-masters

## ■ page-sequence

- Contient les flots de données permettant de remplir des pages
- Fait toujours référence à un page-master ou un page-sequence-master via l'attribut master-reference
- flow
  - Définit le contenu effectif des pages et/ou séquences de pages

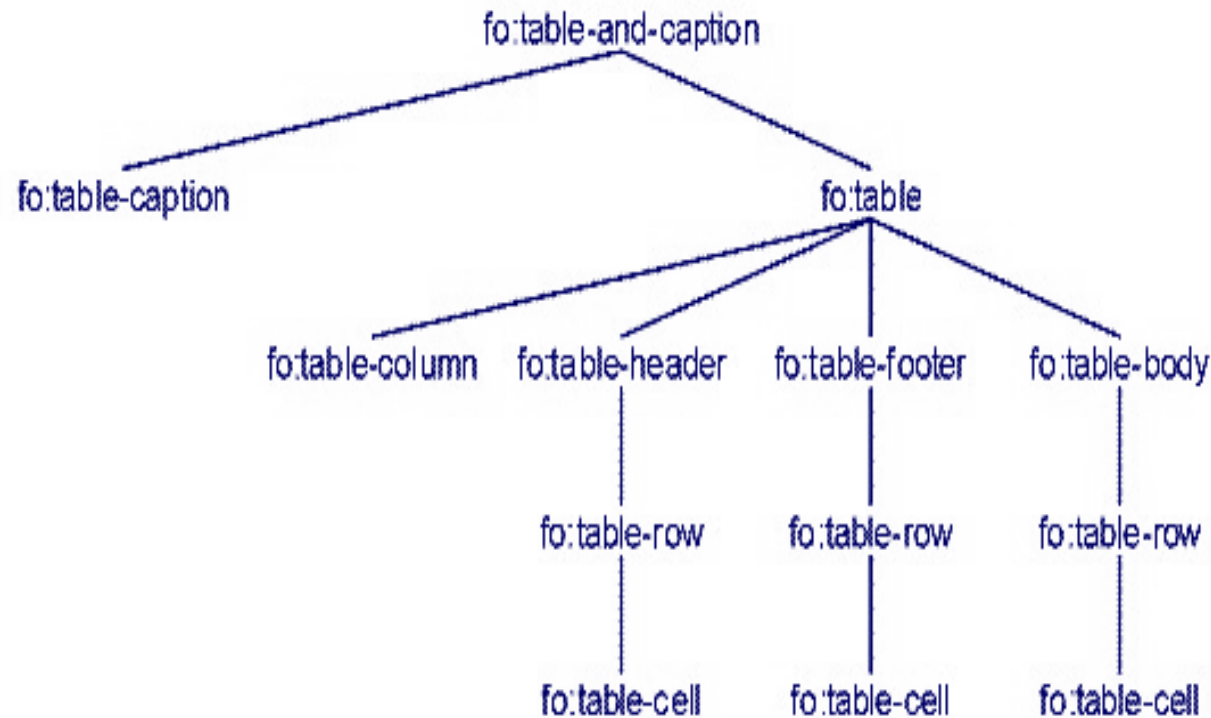
# Les objets de formatage: pagination et layout



# Autres objets de formatage

- inline
  - Texte avec bordure, fond, etc
- table
  - Tables
- list-block, list-item, list-item-body, list-item-label
  - Listes
- basic-link
  - Lien hypertexte
- external-graphics
  - Images
- ...

# Autres objets de formatage: les tableaux



# Implémentations de XSLFO

## ■ Nombre de systèmes commerciaux

- E3 (Arbortext, Inc., <http://www.arbortext.com>)
- Unicorn Formatting Objects (Unicorn Enterprises SA, <http://www.unicorn-enterprises.com>)
- XML2PDF (Altsoft, <http://www.alt-soft.com/>)
- XEP (RenderX, <http://www.renderx.com>)
- Xinc (Lunasil LTD, <http://www.lunasil.com/>)

## ■ Apache FOP

- Apache XML Project (<http://xml.apache.org/fop/>)

# Apache FOP

- Formatting Object Processor
- Fait partie du projet Apache XML
- Pionnier en tant que formateur pour impression basé sur XSLFO
- Écrit totalement en java

## Entrées/sorties de FOP

- FOP lit en entrée un arbre FO
- Le transforme en sorti en des pages sous différents formats
  - PDF , SVG
  - XML , PCL
  - PS , TXT
  - MIF , AWT
- NB: le premier format de restitution est le PDF



## Composition de FOP #1

- FOP inclus dans son package toutes les librairies pour une installation de base du moteur
  - Répertoire fop/lib
  - Fichiers .jar

# Composition de FOP #2

## ■ Xerces & Xalan

- Xerces (Analyseur XML)
- Xalan (processeur XSLT)
- Ecrits en java ( C++, une version Perl de Xerces)
- La combinaison XML-JAVA est multi-plateforme

## ■ Avalon

- Un framework java qui fournit un ensemble d'outils pour la programmation avec les designs patterns

## ■ Batik

- Une composante java pour la manipulation d'images vectorielles SVG (Scalable Vectoriel Graphics)

## ■ XML-API

- Un ensemble de bibliothèques en java pour la manipulation de documents XML

# Installation de FOP

## ■ OS

- ☐ Linux, Win ,Mac

## ■ Pré-requis

- ☐ Java1.2 ou + (Pour Embedded)

- ☐ Xerces, Xalan, batik

- ☐ Optionnellement

  - Bibliothèques graphiques

  - PDF encryption

    - ☐ Pour gérer des restrictions

# Modes d'utilisation de FOP

- Application standalone
- Embedded

## Utilisation de FOP en mode standalone #1

- On peut utiliser FOP en mode batch sous Windows shell sous Unix/Linux
  - fop.bat
  - fop.sh
- Définir la variable JAVA\_HOME
- Taper fop pour avoir les options d'utilisation en mode standalone

## Utilisation de FOP en mode standalone #2

### ■ Exemple d'utilisation

□ `fop foo.fo foo.pdf`

- Génère le fichier pdf à partir du .fo

□ `fop -fo foo.fo -pdf foo.pdf`

- Idem

□ `fop -xsl foo.xsl -xml foo.xml -pdf foo.pdf`

- Génère le fichier pdf à partir du fichier .fo généré par le fichier .xsl

## Utilisation de FOP en mode standalone #3

- Il faut noter que le noyau de FOP ne prend en charge que la transformation du fichier .FO en un autre document .PDF, .TIF, ...
- La transformation de XML vers FO est prise en charge par le processeur XSLT Xalan
- Donc l'utilisateur est le seul responsable face à l'utilisation d'un mauvais document .FO

## Utilisation de FOP en mode embedded #1

- Créer une instance de la classe

  - `org.apache.fop.apps.Driver`

- Ensuite utiliser ses méthodes pour gérer les flux d'entrée (xml, xsl, fo) et le flux de sortie (fo, pdf, ...)



## Utilisation de FOP en mode embedded #2

### ■ Exemple basique

```
import org.apache.fop.apps.Driver;
```

```
...
```

```
Driver driver = new Driver(new InputSource(args[0]),  
new FileOutputStream(args[1]));
```

```
driver.setRenderer(Driver.RENDER_PDF);
```

```
driver.run();
```

### ■ On peut faire plus complexe

- ☐ Génération intermédiaire du fichier FO
- ☐ Logging de tout le processus de génération
- ☐ Serveur de génération de fichier PDF, TXT, ...

## Utilisation de FOP en mode embedded #3

### ■ Servlet

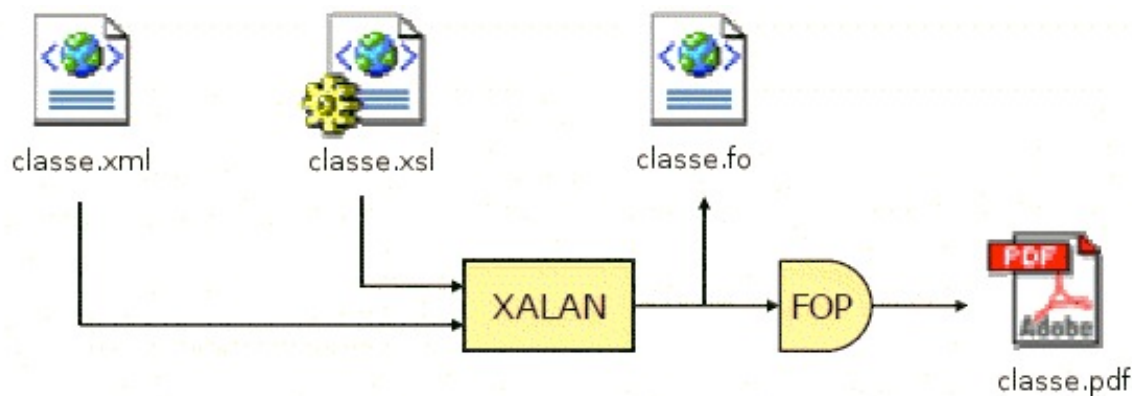
- ☐ Connaître embedded FOP
- ☐ Savoir instancier une servlet

### ■ Ant Task

- ☐ Projet Apache Ant

# Exemple de génération de PDF #1

- Ecrire un fichier XML
- Un fichier XSL
- Utiliser FOP en mode batch pour générer le fichier PDF



## Exemple de génération de PDF #2

- Fichier classe.xml qui contient

- Racine classe

- Nom

- Personne (plusieurs occurrence)

- Nom

- Prénom

- Date\_Naissance

- Email

- fonction

# Exemple de génération de PDF #3

## ■ Fichier XSLT

- Voir le fichier classe.xsl

- A la rencontre de « / »

- Creation de la page maître

- Creation de la page sequence

- Creation flow

- Creation block pour le titre de la table

- Creation block dans lequel on mettra une table

- Définition de la table ainsi que ces propriétés

- Definition d'un template pour le remplissage des éléments de la table