



Université Ibn Zohr
Faculté des sciences – Département Informatique

XML eXtensible Markup Language

Ayoub SABRAOUI

Master OTI –S1

Année universitaire 2016/2017

1

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

- DOM (Document Object Model)
- XPath (Chemins d'accès au arbre XML)
- Transformations XSLT

2

Objectifs

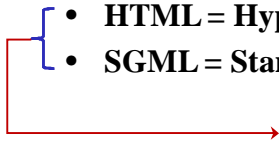
■ On veut représenter des données

- 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**

3

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.....

4

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

5

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 !**

6

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/>

7

XML, qu'est-ce que c'est ?

- balises descriptives (signification des données) plutôt que procédurales (présentation des données)
- libre, indépendant des plateformes logicielles ou matérielles
- XML est extensible: ne contient pas un ensemble fixe de balises
- les documents XML doivent être bien formés suivant une syntaxe définie, et peuvent donc être formellement validés
- XML est particulièrement adapté à l'échange de données et de documents.

8

XML, qu'est-ce que c'est ?

Parsers et Décodage des documents XML

- L'extraction des données d'un document XML se fait à l'aide d'un outil appelé analyseur syntaxique (en anglais **parser**, ou parseur) qui permet :
 - d'extraire les données d'un document XML (analyse du document ou parsing)
 - éventuellement, de vérifier la validité du document.

9

DEFINITION

- eXtensible Markup Language
 - Recommandation (norme) du W3C
 - Spécifiant un langage
 - Constitué d'un ensemble d'éléments appelés balises
 - Utilisable pour créer d'autres langages
- 2 concepts fondamentaux
 - Structure et présentation sont séparés
 - Les balises ne sont pas figées

10

DEFINITION

■ Conséquences :

- XML est un format de document
- XML est un format de données (dialectes)
- XML est un méta-langage (une famille de langages)

■ En simplifié :

- « XML est un langage de description de documents structurés » (www.w3c.org/XML).

11

INTERÊT de XML

■ Richesse sémantique

- Dédié au traitement des données
- Soutenant une grande variété d'applications

■ Facilité de mise en œuvre

- Simple et lisible
- Portable et facilement utilisable sur Internet
- Assurant un développement aisé

12

Structure de documents XML

■ Prologue :

- Rôle équivalent au <HEAD> HTML,
- Meta-Information : {
 - **Instructions** de traitement
 - **commentaires**
(non interprétable par le parseur)

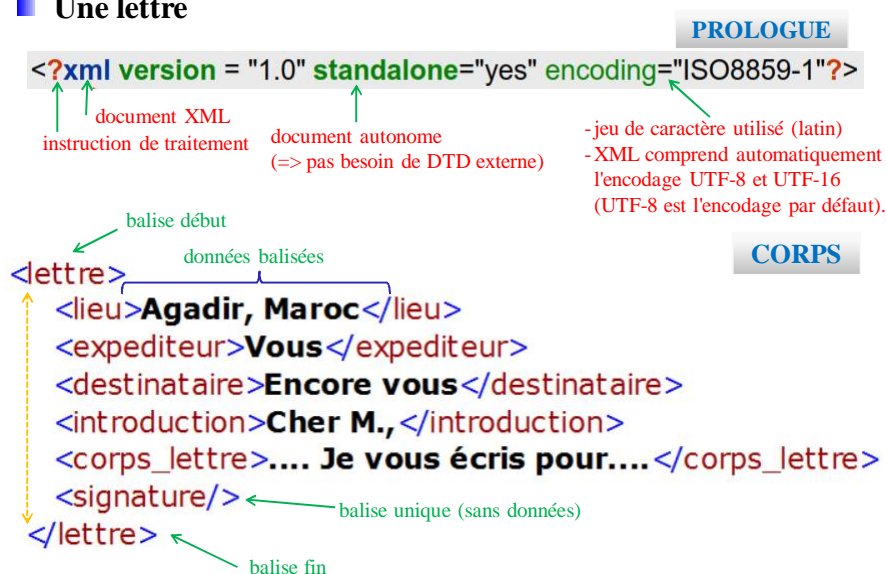
■ Corps :

- Rôle équivalent au <BODY> HTML
- Les données formatées: (structure arborescente) {
 - Balises d'encadrement
 - **Attributs associés** aux balises
 - **Données encadrées** par les balises

13

Exemple XML

■ Une lettre



Prologue d'un document XML

■ Exemple

document XML 1.0

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

```
<?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")

15

Corps d'un document XML

■ Exemple

```
<liste_CD>
  <CD>
    <artiste type="individuel">Artiste un</artiste>
    <titre no_pistes="4">titre un</titre>
    <pistes>
      <piste>piste 1</piste>
      <piste>piste 2</piste>
    </pistes>
    <prix devise="dirham" paiement="CB">30.00</prix>
    <en_vente/>
  </CD>
  <CD>.....</CD>
</liste_CD>
```

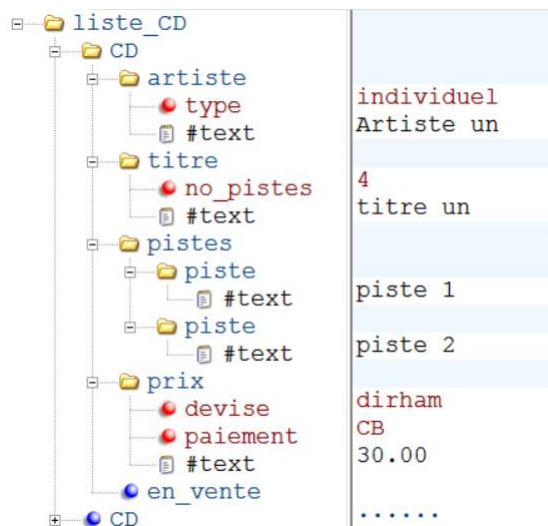
Diagram illustrating the structure of an XML document body with numbered annotations:

- 1: Start of the root element `<liste_CD>`
- 2: End of the root element `</liste_CD>`
- 3: Start of a child element `<CD>`
- 4: End of a child element `</CD>`
- 5: Start of an attribute value `type="individuel"`
- 6: End of an attribute value `type="individuel"`
- 7: End of an attribute value `paiement="CB"`
- 8: Start of an attribute value `devise="dirham"`
- 9: Start of a child element `<pistes>`

16

Corps d'un document XML

■ Arbre des balises sur l'exemple



17

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) contiennent des **attributs (5)(8)**,

18

Structure des documents XML

Synthèse

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

- Balises du prologue :

`<?nom_balise_traitement ...?>`

Déclaration

`<!DOCTYPE ...>`

Type de document

- 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/>`

XML : les commentaires

- en XML les commentaires se notent :

`<!-- texte du commentaire -->`

- Les contraintes d'utilisation sont

- pas de double tirets dans le texte,
- pas de commentaire dans un élément (l'exemple ci-dessous est incorrect),

`<produit
 nom="DVD"
 prix='100' <!-- en euros -->
>`

- les commentaires sont ignorés (plus ou moins),

XML : les balises (éléments)

■ Forme générale :

`<nom_d_élément> contenu </nom_d_élément>`

■ Les noms sont libres (contrairement à HTML). Ils obéissent à quelques règles:

- 1er caractère { alphabétique, «-», «_» },
- les autres caractères { alphabétique, chiffre, «-», «_», «:» }.
- pas de blanc,
- «xml» au début est interdit (maj./min.).

■ La balise de fermeture est obligatoire.

21

XML : les balises (éléments)

■ Le contenu d'un élément peut être

- vide (`<toc></toc>` ou `<toc/>`),
- du texte (sauf «<» et «&») basé sur l'encodage,
- un ou plusieurs éléments complets
`<toc> ... </toc>`
- une répétition de textes et d'éléments,
`<article> Le langage <def>XML</def> contient <liste>`
`<élément> du texte, </élément>`
`<élément> des éléments, </élément>`
`</liste></article>`
- Les blancs comptent: `<a> X ` est différent de `<a>X`.

22

XML : arbre d'éléments

- Un document XML est un et un seul arbre d'éléments. C'est à dire :

- Pas de chevauchement d'éléments. La notation suivante :

```
<list> ... <item> ... </list> ... </item>
```

est invalide. Il faut la corriger comme suit

```
<list> ... <item> ... </item> ... </list>
```

- Un document XML est composé d'un seul élément. La notation suivante :

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

```
<article> ... </article>
```

```
<article> ... </article>
```

est invalide. Il faut la corriger comme suit

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

```
<stock>
```

```
  <article> ... </article>
```

```
  <article> ... </article>
```

```
</stock>
```

23

XML : les attributs

- Un élément ouvrant peut être enrichi par des couples de la forme **attribut1="valeur1"** comme dans l'exemple

```
<produit nom="DVD" prix='200'>
```

- Syntaxe : **nom='valeur' ou nom="valeur"**

- La valeur doit être entourée d'apostrophes si elle contient des guillemets, et inversement.

- Caractères interdits : ^, % et &

- Le nom des attributs suit les mêmes règles syntaxiques que les noms d'éléments.

- Attributs comme ci-dessus ou sous-éléments ?

```
<produit>
```

```
  <nom>DVD</nom>  <prix>150</prix>
```

```
</produit>
```

- L'attribut doit changer l'interprétation des données:

```
<prix monnaie="Euro"> 150 </prix>
```

24

XML : les attributs réservés

- `xml:lang='langue'` permet de définir la langue utilisée dans l'élément et tous les sous-éléments.

La langue suit la norme ISO 3166 définie par la RFC 1766 (Request For Comment). Par exemple `fr` ou `en-US` ou `fr-FR`.

- `xml:space='preserve'` ou `xml:space='default'` permet de définir l'interprétation des espaces dans l'élément et tous les sous-éléments.
- `xml:id='identificateur'` permet d'associer une et une seule clef à un élément .
- `xml:idref='identificateur'` permet de faire référence à une clef.

```
<section id='intro'>
  <titre>introduction à XML</titre>
  ... </section>

<section>
  <p> après la section
  <xref idref='intro'>d'introduction</xref>
  nous allons passer au plat de résistance...
</section>
```

25

XML : les références d'entités

- Les entités sont des fragments de document XML définis dans la DTD. La référence d'entité se note :

`&nom_de_l_entité;`

- Il existe des entités prédéfinies :

```
* &amp; donne &
* &lt; donne <
* &gt; donne >
* &quot; donne «
* &apos; donne '
* &#nnn; donne le caractère de code décimal nnn,
* &#xnnn; donne le caractère de code hexadécimal nnn,
```

- Un exemple :

```
<texte> en HTML, la balise
&lt;p&gt; est très utile !
&#169;
</texte>
```

26

XML : section littérale

- Avec les sections littérales Il est possible de stopper l'interprétation des caractères spéciaux. La syntaxe est la suivante :

`<![CDATA[texte non soumis à l'analyse]]>`

L'exemple précédent devient

`<texte><![CDATA[en HTML, la balise
<p> est très utile !]]>
© </texte>`

27

XML : espaces de noms

- Un problème apparaît si on mélange deux textes XML dont les éléments ont le même nom. Par exemple

```
<produit>
  <nom>...</nom>
  <desc>...</desc>
</produit>

<fournisseur> <nom>...</nom>
  <desc> <adr>...</adr>
  <tél>...</tél> </desc>
</fournisseur>
```

- Pour régler ce problème on enrichit le nom de l'élément :

```
<fsa:produit
  xmlns:fsa=http://www.fsa.univ-ibn-zohr.ma>
  <fsa:nom>...</fsa:nom>
  <fsa:desc>...</fsa:desc>
</fsa:produit>
```

28

XML : espaces de noms

- Attention, le préfixe n'est qu'une macro. C'est l'espace de nom qui compte. Les deux éléments suivants sont les mêmes:

```
<fsa:produit
xmlns:fsa=http://www.fsa.univ-ibn-zohr.ma>
... </fsa:produit>
```

```
<inf:produit
xmlns:inf=http://www.fsa.univ-ibn-zohr.ma>
... </inf:produit>
```

- Les espaces de noms doivent être utilisés si le document XML rédigé est destiné à être mélangé à d'autres sources.
- On peut fixer l'espace de noms par défaut avec la syntaxe:

```
<produit xmlns=http://www.fsa.univ-ibn-zohr.ma>
  <nom>...</nom> <desc>...</desc> </produit>
</produit>
```

cela évite d'utiliser le préfixe.

29

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 fermante
- 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

30

Un 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

31

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

- Alors

- Le document XML peut être échangé ! (format standardisé)

32

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

- DOM (Document Object Model)
- XPath (Chemins d'accès au arbre XML)
- Transformations XSLT

33

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)

34

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

35

Élément et attribut

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

36

Elément et attribut

■ Nature des attributs : optionnels, obligatoires, valeur déterminée

- optionnel sans valeur par défaut
`<!ATTLIST personne att1 CDATA #IMPLIED>`
- optionnel avec valeur par défaut
`<!ATTLIST personne att1 "bidule">`
- obligatoire
`<!ATTLIST personne att1 CDATA #REQUIRED>`
- fixe
`<!ATTLIST personne att1 CDATA #FIXED "bidule">`

■ **Exemple :**

```
<!ATTLIST personne id ID #REQUIRED>
<!ATTLIST personne att1 CDATA #IMPLIED att2 CDATA #IMPLIED>
```

37

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+*)

38

Types de données

- **CDATA** : Données brutes qui ne seront pas analysées par le parseur (*Unparsed*) *Character DATA*
- **PCDATA** : Données de type texte dans l'encodage courant *Parsable Character DATA*
- **Enumération** : Liste de valeurs séparées par « | » (oui | non | peut-etre)
- **ID** : Identifiant pour l'élément (doit être unique dans le document)
- **IDREF, IDREFS** : Référence à un ID de ce document (resp. plusieurs références séparées par des espaces)
- **ENTITY, ENTITIES** : La valeur de l'attribut doit être le nom d'une entité déclarée dans la DTD (resp. un ensemble d'entités séparées par des espaces)
- **ANY** : Tout texte possible
- **EMPTY** : Vide

39

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>

```

40

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>70 000 Dirhams</PRIX>
  </VOITURES>
  .....
</CATALOGUE>
```

41

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">Fatima</PERSONNE>
  <PERSONNE PNUM = "P2">Mohamed</PERSONNE>
  <PERSONNE PNUM = "P3" MERE="P1" PERE="P2">Ali</PERSONNE>
  <PERSONNE PNUM = "P4" MERE="P1">Hajar</PERSONNE>
</DOCUMENT>
```

42

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">
...
```

43

Entités dans les DTD

■ Entité

- Permet la réutilisation dans une DTD

Syntaxe

- 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>
```

44

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 alternatifs** d'éléments → (a|b|c)
 - **Mixte hiérarchique** → (a,(b|c),d)
- ✓ indicateurs d'occurrences :
 - ? (zéro ou une),
 - * (zéro ou plusieurs)
 - + (une ou plusieurs)

45

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.

46

Insuffisance des DTD

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

47

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 nommage

48

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

49

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">
    //...
    //...
</xs:schema>
      
```

50

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>Ahmed</to>
  <from>Khalid</from>
  <heading>Rappel</heading>
  <body>Rendez-vous demain</body>
</note>
```

51

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)

53

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.

54

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

55

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"/>
```

57

Eléments Complexes

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

- 3 types d'éléments complexes:

- élément qui contient d'autres éléments et/ou des attributs
- 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!

58

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>Ahmed</firstname>
  <lastname>RAKI</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>
```

59

Les types complexes

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

Document XML

```
<employee>
  <firstname>Ahmed</firstname>
  <lastname>RAKI</lastname>
</employee>
```

Schéma XML correspondant :

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

60

Les types complexes

■ 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>
```

61

Les types complexes

■ 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>
```

62

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>
```

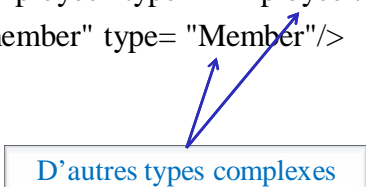
64

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

65

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"
        minOccurs="0" maxOccurs="10" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

66

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="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="ISBN">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d-\d{3}-\d{5}-\d"/>
  </xsd:restriction>
</xsd:simpleType>
```

Un élément **isbn** déclaré du type **ISBN** défini ci-dessus pourrait avoir le contenu suivant :

```
<isbn>2-311-01400-6</isbn>
```

Héritage de types (simple)

■ Autres exemples de restrictions

Exemple: énumération

```
<xs:attribute name="type" use="required">
  <xs:simpleType>
    <xs:restriction base="xs:token">
      <xs:enumeration value="domicile"/>
      <xs:enumeration value="bureau"/>
      <xs:enumeration value="mobile"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
```

Exemple: restriction de la valeur d'un élément simple

```
<xsd:element name="quantity">
  <xsd:simpleType>
    <xsd:restriction base="xsd:positiveInteger">
      <xsd:maxExclusive value="100"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

69

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>
```

70

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

71

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>

```

ref est une référence à l'élément **comment**

Les deux attributs **name** et **ref** ne peuvent pas être présents simultanément dans l'élément `xsd:element`.

XML Schema : exemple (2)

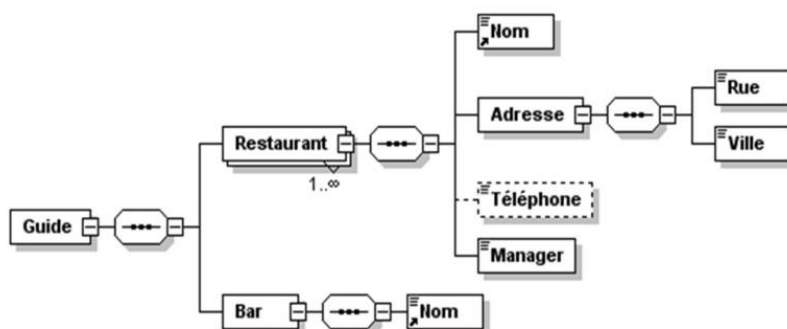
```

<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>

```

73

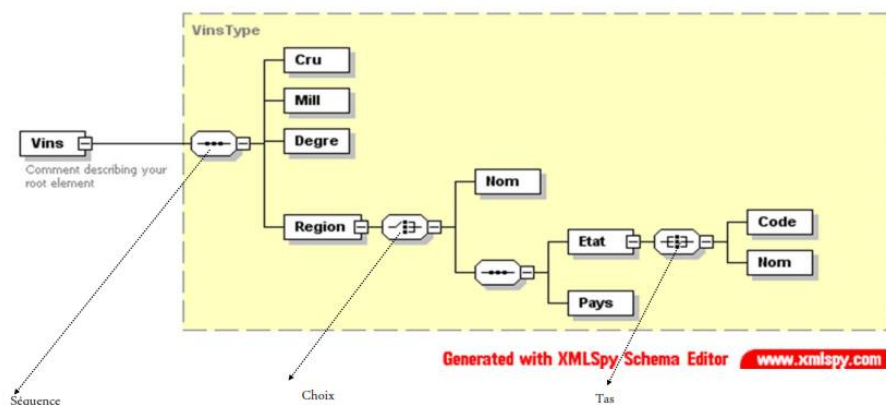
Diagramme XML Spy



Generated with XMLSpy Schema Editor www.xmlspy.com

74

Diagramme de type (XML Spy)



75

L'inclusion d'un Schéma

- **Objectif:** injecter un schéma dans un autre=> réutilisation
- L'élément **"include"** permet de réaliser cette inclusion
 - Se positionne sous la racine du schéma.
 - Le schéma inclus **épouse** l'espace de noms du schéma principal

■ Exemple:

Soit le schéma suivant **personne.xsd** :

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="personneType">
    <xs:sequence>
      <xs:element name="nom" type="xs:string"/>
      <xs:element name="prenom" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Le schéma **employe.xsd** inclut le schéma **personne.xsd** :

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:include schemaLocation="personne.xsd"/>
  <xs:element name="employe" type="personneType"/>
</xs:schema>
```

Exemple d'un XML instance du schéma :

```
<employe xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="employe.xsd">
  <nom>NOM</nom>
  <prenom>Prenom</prenom>
</employe>
```

L'inclusion et la redéfinition d'un Schéma

- **Objectif:** même chose que l'inclusion + la possibilité de redéfinir un type simple ou complexe
- équivalent de la surcharge en OO
 - L'élément "**redefine**" permet de réaliser cette inclusion
 - se positionne sous la racine du schéma.
 - Le schéma inclus **épouse** l'espace de noms du schéma principal
- **Exemple**

Le schéma employe.xsd:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:include schemaLocation="personne.xsd"/>
  <xs:element name="employe" type="personneType"/>
</xs:schema>
```

Exemple d'un XML instance du schéma employe2.xsd :

```
<employe xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="employe2.xsd">
  <nom>NOM</nom>
  <prenom>Prenom</prenom>
  <telephone>999</telephone>
</employe>
```

Le schéma employe2.xsd redefine employe.xsd:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:redefine schemaLocation="employe.xsd">
    <xs:complexType name="personneType">
      <xs:complexContent>
        <xs:extension base="personneType">
          <xs:sequence>
            <xs:element name="telephone" type="xs:string"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:redefine>
</xs:schema>
```

77

Key et Keyref

- Même chose que ID et IDREF avec plus de précision
 - Permet de préciser pour un IDREF, quel est l'ID exacte vers lequel il pointe

```
<root
  xsi:noNamespaceSchemaLocation="ExempleKey.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <AAA>
    <a id="x"/>
    <a id="y"/>
  </AAA>
  <BBB>
    <b idref="x"/>
    <b idref="y"/>
    <b idref="y"/>
  </BBB>
</root>
```

```
<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" >
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence minOccurs="1" maxOccurs="1">
        <xs:element name="AAA" type="myAAA"/>
        <xs:element name="BBB" type="myBBB"/>
      </xs:sequence>
    </xs:complexType>
    <xs:key name="myId">
      <xs:selector xpath="/AAA/a"/>
      <xs:field xpath="@id"/>
    </xs:key>
    <xs:keyref name="myIdref" refer="myId">
      <xs:selector xpath="/BBB/b"/>
      <xs:field xpath="@idref"/>
    </xs:keyref>
  </xs:element>

  <xs:complexType name="myAAA">
    <xs:sequence minOccurs="1">
      <xs:element name="a" minOccurs="1" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="id" type="xsd:NCName" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="myBBB">
    <xs:sequence minOccurs="1">
      <xs:element name="b" minOccurs="1" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="idref" type="xsd:NCName" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

78

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>
```

79

Quelques outils de conception

Editeur	Outil	Support
Tibco	Turbo XML 2.0	DTD Schéma
Altova	XML Spy 4.0	DTD Schéma
SyncRO Ltd.	Oxygen 2.0	DTD Schéma
Data Junction	XML Junction	Schéma
Insight Soft.	XMLMate 2.0	DTD Schéma
Microstar Soft.	Near & Far Designer	DTD

80

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 ?

81

Bilan Schéma

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

82

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 formés (*syntaxe conforme à XML*)
 - valides (*structure conforme à une grammaire*)

- Outils et standards de manipulation pour les documents

83

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**
 - DOM (Document Object Model)
 - XPath (Chemins d'accès au arbre XML)
 - Transformations XSLT

84

Traitement des documents XML

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

85

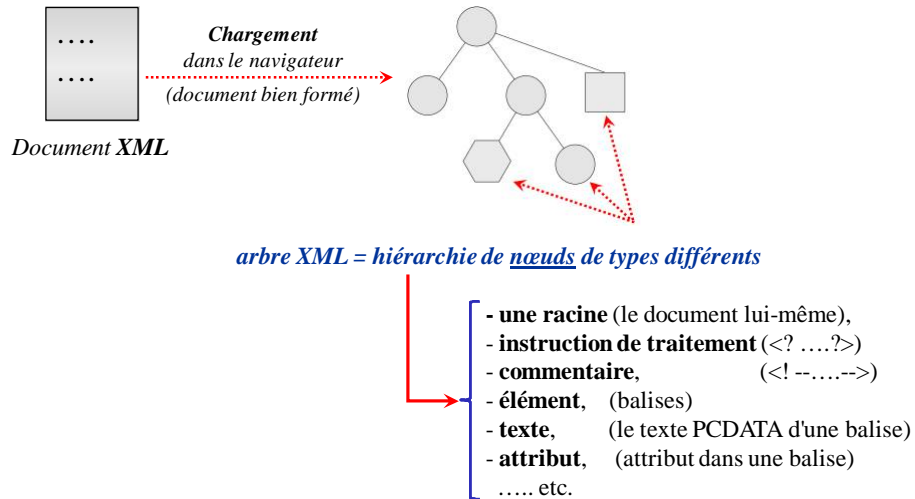
Objectifs

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

- Récupérer {
 - Les valeurs des **balises**
 - Les valeurs des **attributs**
 - Des **fragments d'un document XML**
- Par rapport à {
 - Leur **type** (instruction de traitement, commentaire, balise donnée ou attribut)
 - Leur **nom** (dans le cas des balises et attributs)
 - Leur **position** (avant, après, à coté d'une autre balise...)
- Dans le but de {
 - **Afficher leur contenu** (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)

86

Construction de l'arbre XML



87

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>
```

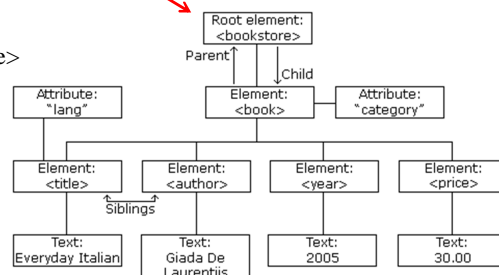
```
    <year>2005</year>
```

```
    <price>29.99</price>
```

```
  </book>
```

```
  ...
```

```
</bookstore>
```

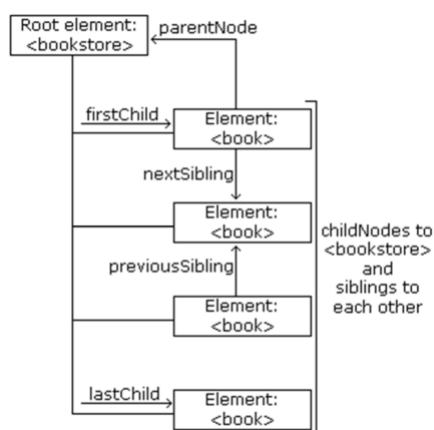


88

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

- On peut naviguer entre les nœuds en utilisant les différentes relations qui peuvent exister entre eux:

- parentNode
- childNodes
- firstChild
- lastChild
- nextSibling
- previousSibling



89

Structure de l'arbre XML en Nœuds

- 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

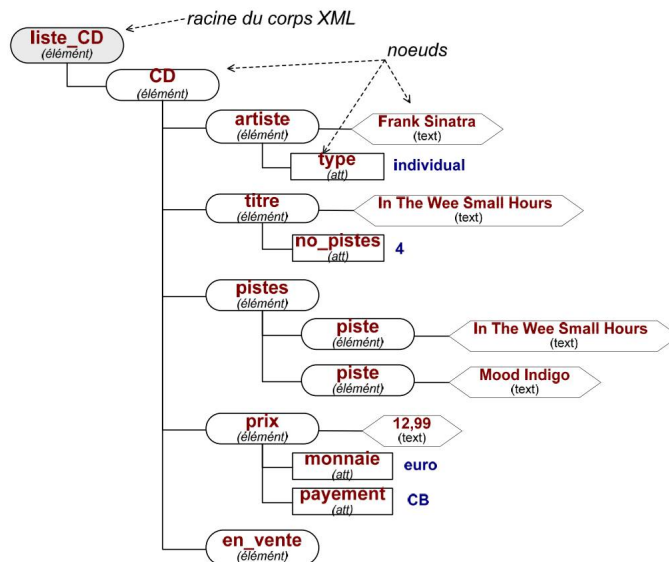
- Les nœuds fils d'une "racine" sont de type :

- "**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 !

90

Exemple d'arbre XML



91

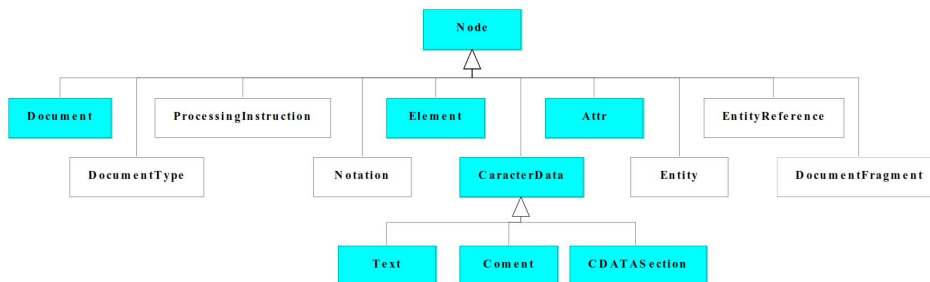
Parcours des arbres XML en utilisant DOM (Document Object Model)

- 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** !

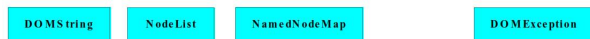
92

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

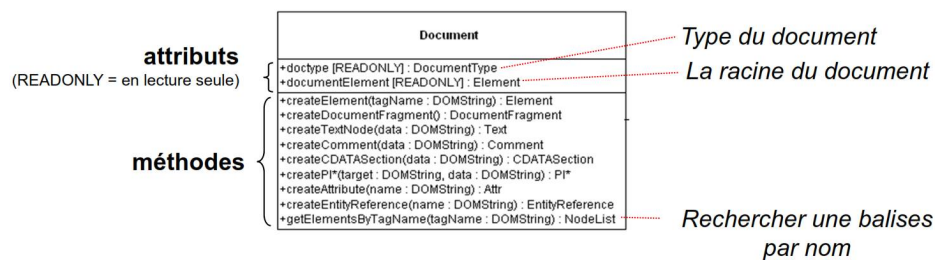
Autres nœuds :

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

94

La classe *Document*

- Représente le document XML



- Syntaxe des :

- **attributs** : *nom_de_l'attribut* [accès] : *type_de_l'attribut*

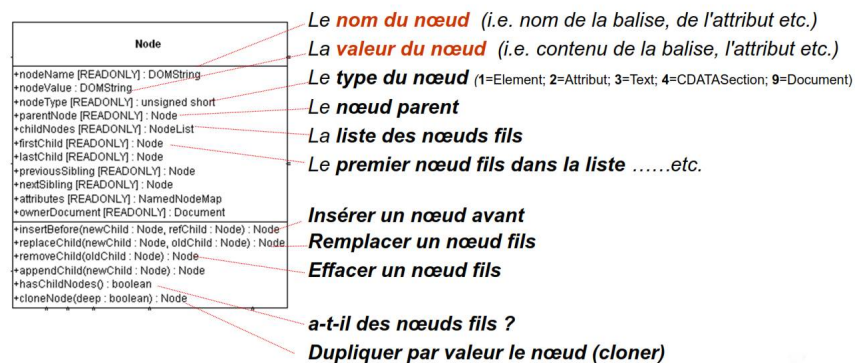
- **méthodes** : *nom_de_la_méthode_publique* (paramètres) : *type_de_la_valeur_de_retour*

$$\underbrace{\hspace{10em}}_{\text{nom_parametre : type_parametre}}$$

95

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 !



96

La classe *Node* (2)

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

Type de Noeud	nodeName	nodeValue	attributs
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

97

La classe *Element*

- Représente une **balise** XML

Element
*tagName [READONLY] : DOMString
*getAttribute() : DOMString
*setAttribute(name : DOMString, value : DOMString) : void
*removeAttribute(name : DOMString) : void
*getAttributeNode(name : DOMString) : Attr
*setAttribute(newAttr : Attr) : Attr
*removeAttributeNode(oldAttr : Attr) : Attr
*getElementsByTagName(name : DOMString) : NodeList
*normalize() : void

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

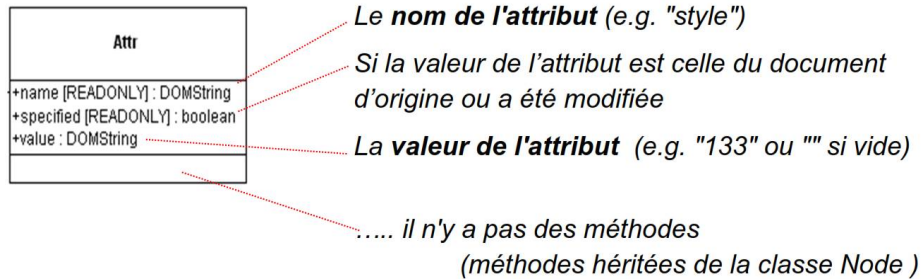
Effacer un attribut

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

98

La classe *Attr* (attribut)

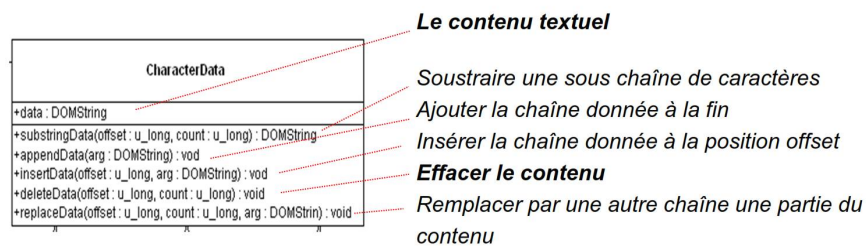
- Représente un attribut d'une balise XML



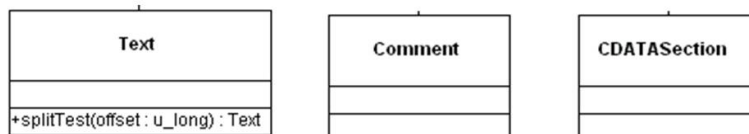
99

La classe *CharacterData*

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



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



Classes DOM auxiliaires

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

NodeList
+length [READONLY] : u_long
+item(index : u_long) : Node

Nombre d'éléments dans la liste de nœuds

Le nœud avec l'index ...

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

NamedNodeMap
+length [READONLY] : u_long
+getNamedItem(name : DOMString) : Node
+setNamedItem(arg : Node) : Node
+removeNamedItem(name : DOMString) : Node
+item(index : u_long) : Node

Nombre d'éléments dans la liste de nœuds

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)

101

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)

102

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

1. Importer les packages 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(new File(file));
```

103

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-éléments

```
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 nœuds de différents types et ne peuvent donc pas être directement castés en Element

104

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 ?

105

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

106

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();
    }
}

```

107

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

108

Conclusion DOM

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

109

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/>

110