Introduction au langage XML
2: XML schéma et Xpath

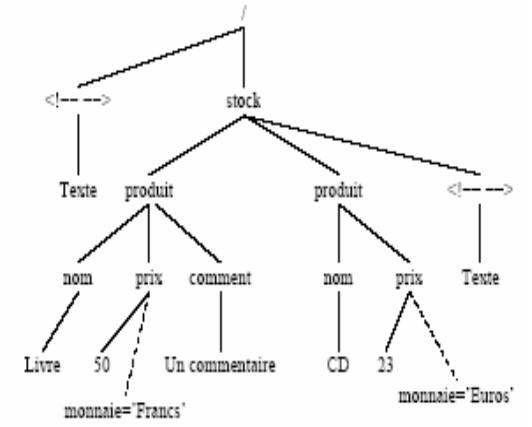
XPath : Introduction

- Le langage XPATH offre un moyen d'identifier un ensemble de noeuds dans un document XML.
- Toutes les applications ayant besoin de repérer un fragment de document XML peuvent utiliser ce langage.
- Les feuilles de style XSL, les pointers XPOINTER et les liens XLINK utilisent de manière intensive les expressions XPATH.
- XPATH est un premier pas vers un langage d'interrogation d'une base de données XML (XQL et XML-QL).

XPath : Principe

Expression XPath : Chaîne spécifiant la sélection d'un ensemble de noeuds dans un arbre DOM

```
<!-- Texte -->
<stock>
<produit>
<nom> Livre </nom>
<prix monnaie="Francs"> 50 </prix>
<comment> Un commentaire </comment>
</produit>
<produit>
<nom> CD </nom>
<prix monnaie="Euros"> 23 </prix>
</produit>
<!-- Texte -->
</stock>
```



3

XPath : Expressions

- Une *expression XPath* désigne un ou plusieurs noeuds en exprimant des *chemins* dans un arbre DOM du document XML
- L'*évaluation* d'une expression donne un résultat qui peut être soit une valeur numérique ou alphanumérique, soit un sous-ensemble des noeuds de l'arbre
- Enfin cette évaluation tient compte d'un *context général* qui détermine le résultat.

4

XPath : Expressions (2)

- La forme générale d'une expression XPATH est
  - selecteur1/selecteur2/... (expression relative)
  - /selecteur1/selecteur2/... (expression absolue)
- Chaque sélecteur sélectionne un ensemble de noeuds en fonction du résultat du sélecteur précédent
- L'ensemble initial est soit le noeud courant (forme relative) soit la racine (forme absolue).
- Exemple : /Stock/Produit/Comment

5

XPath : Sélecteur de nœud

- Les sélecteurs de noeuds sont de la forme :
  - axe::filtre[condition]
  - axe::filtre
  - filtre[condition]
  - filtre
- *L'axe* qui définit le sens de parcours des noeuds à partir du noeud contexte ;
- *Le filtre* qui indique le type des noeuds qui seront retenus dans l'évaluation de l'expression ;
- *La (ou les) condition(s)* qui exprime(nt) des propriétés que doivent satisfaire les noeuds retenus à l'issue du filtrage pour être finalement inclus dans le résultat.

6

XPath : Sélecteur de nœud

- Comportement : un sélecteur de noeuds retient les noeuds qui
  - sont dans l'axe (ou sont fils directs du nœud courant si l'axe est omis),
  - ont une étiquette qui respecte le filtre,
  - respectent la condition.
- Exemple :
  - /Stock/Descendant::Prix/attribute:: monnaie

7

XPath : Axes en avant

- Les axes qui permettent de descendre dans l'arbre :
  - Self le noeud courant,
  - Child les fils du noeud courant (c'est l'axe par defaut),
  - Descendant les descendants du noeud courant
    - Exemple /stock/descendant::prix
  - descendant-or-self les descendants du noeud courant plus lui même.

8

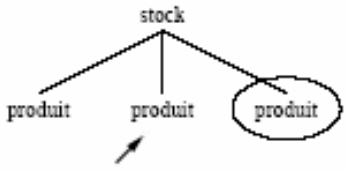
XPath : Axes en arrière

- Les axes qui permettent de remonter dans l'arbre:
  - Parent — le noeud parent du noeud courant,
  - Ancestor — les ascendants du noeud courant (dans l'exemple ci-dessous le noeud courant est le prix) :
    - Exemple — ancestor::produit/prix
  - ancestor-or-self — que dire ?

9

XPath : Axes à droite et à gauche

- Autres formes pour les axes :
  - following-sibling — les noeuds frères placés après le noeud courant (à droite)

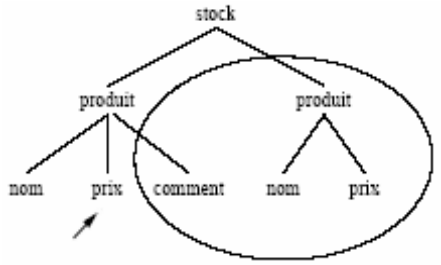


  - preceding-sibling — les noeuds frères placés avant le noeud courant (à gauche)

10

XPath : Axes avant et après

- Autres formes pour les axes :
  - Following — tous les noeuds placés après dans le document,



  - Preceding — tous les noeuds places avant dans le document

11

XPath : Axes avec type

- Autres formes pour les axes :
  - Attribute — les noeuds de type attribut du noeud courant,
  - Namespace — les noeuds de type espace de nom du nœud courant,

12

XPath : Axes - résumé

Axe	Description
child	les fils du noeud contexte (axe par défaut)
attribute	les attributs du noeud contexte
parent	le père du noeud contexte
descendant	tous les descendants du noeud contexte
ancestor	tous les ancêtres du noeud contexte
self	le noeud contexte lui-même
preceding-sibling	tous les frères gauches du noeud contexte
following-sibling	tous les frères droits du noeud contexte
preceding	les noeuds précédant le noeud contexte dans l'ordre de parcours du document
following	les noeuds suivant le noeud contexte dans l'ordre de parcours du document
descendant-or-self	les descendants du noeud contexte, et le noeud contexte lui-même
ancestor-or-self	les ancêtres du noeud contexte, et le noeud contexte lui-même
namespace	s'applique aux espace de noms

13

XPath : Les filtres: introduction

- Un filtre permet d'éliminer un certain nombre de noeuds parmi ceux sélectionnés par un axe.
- Il existe essentiellement deux types de filtrage pour sélectionner des noeuds :
  - *par leur type* et
  - *par leur nom*.
- Ces deux critères de sélection sont les plus utilisés dans les expressions XPath

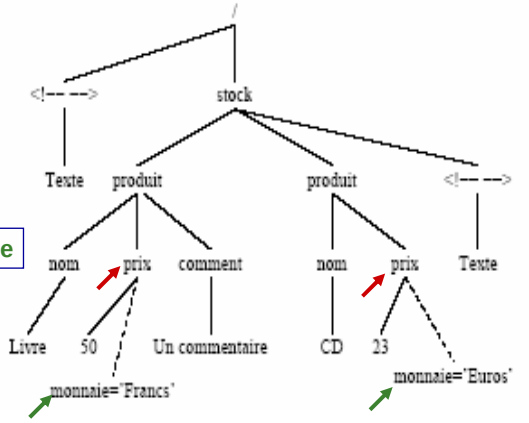
14

XPath : Les filtres par nom

Filtre	Nœuds sélectionnés
nom	tous les noeuds de l'axe qui portent ce nom

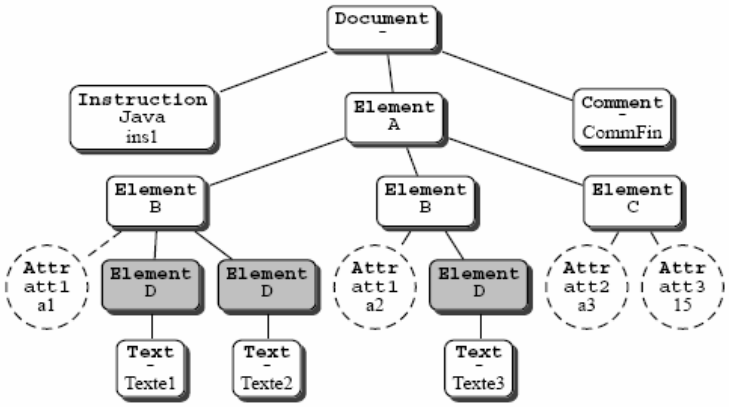
/stock/produit/prix

/stock/produit/prix/attribute::monnaie



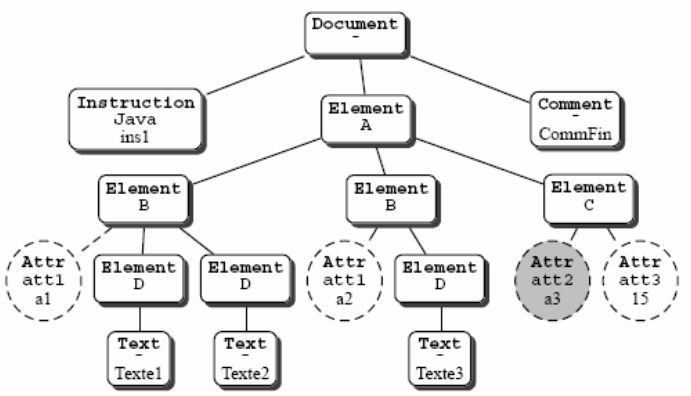
XPath : Exemple de filtre par nom

L'expression **/A/B/D** désigne, en partant de la racine du document, l'élément de nom A, puis les éléments de nom B fils de A, et enfin les éléments de nom D fils de B.



XPath : Exemple de filtre par nom (2)

L'expression **/descendant::node()/@att2** qui, à partir de la racine, va sélectionner tous les noeuds à l'exception des attributs, puis pour chacun de ces noeuds on cherche l'attribut att2.



XPath : Filtres par type de noeuds

Filtre	Nœuds sélectionnés sur l'axe
*	tous les noeuds de type **Element** (avec tous les axes sauf attribute) ou de type **Attr** (avec l'axe attribute)
text()	tous les noeuds de type **Text (ex.** /stock/produit/*/text())
comment()	tous les noeuds de type **Text**
Processing-instruction()	tous les noeuds de type **ProcessingInstruction**
id(label)	Tous les noeuds repère par une étiquette (ex. id('CD')/prix)
node()	tous les noeuds (de n'importe quel type)

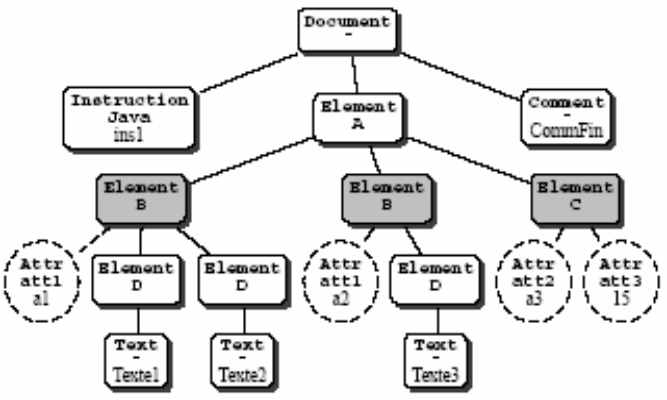
XPath : Exemples de filtres par type

Expression	Signification
/comment()	Tous les commentaires depuis la racine du document
/A/B//text()	tous les noeuds de type Text situés sous un élément B
processing-instruction('java')	sélectionne uniquement les instructions de traitement dont la cible est java
/child::A/child::B/descendant-or-self::node()/text()	Expression complète de /A/B//text()

19

XPath : Exemple de filtre par type

L'expression **/A/*** va sélectionner tous les éléments fils de l'élément racine <A> quelque soit leur nom



20

XPath : Abréviation

Abréviation	Expression complète
.	self::node()
..	parent::node()
@	Attribute::
@att	Attribute::att
@*	Attribute::*
A@*	Child::A/Attribute::*
Axe par défaut	Child::
/A	/Child::A
A/*	Child::A/child::*
Sélecteur par défault	descendant-or-self::node()
//	/descendant-or-self::node()/
.//	self::node()/descendant-or-self::node()/
A//B	child::A/descendant-or-self::node()/child::B

21

XPath : Abréviation(2)

- Noeuds: comme un système de fichier:
  - / = racine
  - commence par / = chemin depuis la racine du doc
  - ne commence pas par / = relatif au noeud courant pour l'application
  - commence par // = sélectionne toutes les sous-parties
  - * = remplace un niveau de hiérarchie
  - * à la fin = tout le contenu
  - .. = remonter
- Attributs: préfixé par @:
  - noeud/@attribut = contenu de l'attribut
  - @* = sélectionne tous les attributs

22

XPath : Abréviation (3)

- Exemple :
  - prix/attribute::monnaie devient prix/@monnaie
  - /descendant-or-self::prix devient //prix
  - produit/self::node() devient produit/
  - prix/parent::node() devient prix/..
  - produit/self::node() devient produit/
  - produit[position() = 4] devient produit[4]

23

XPath : Les prédicats (conditions)

- Il existe plusieurs formes possibles de conditions:
  - axe::filtre[numéro] sélectionne les noeuds en fonction de leur position.
    - Exemple : /stock/produit[3]
  - axe::filtre[expression XPATH] sélectionne les noeuds pour lesquels la sous expression renvoie un ensemble de noeuds non vide.
    - Exemple : /stock/produit[prix/attribute::monnaie]

24

XPath : Les prédicats (2)

- axe::filtre[condition1][condition2]...
- sélectionne les noeuds identiques par filtre ssi les conditions sont respectées.
- Attention, ces deux expressions sont différentes :
  - produit[2][comment]

    sélectionne les noeuds produit en deuxième position qui possède un élément fils comment.
  - produit[comment][2]

    sélectionne le deuxième noeud produit qui possède un élément fils comment.

25

XPath : Les prédicats (3)

- Les conditions peuvent également s'écrire (les relations possibles sont =, =!=, <, <=, >, >=) :
  - valeur1 relation valeur2
  - condition1 and condition2
  - condition1 or condition2
  - not(condition)
  - true()
  - false()
  - boolean(objet)
- Opérations sur les nombres : +, -, div, mod

26

XPath : fonctions

- On peut utiliser des fonctions dans les expressions. Un ensemble de fonctions sont prédéfinies.
- Elles sont classées par catégories:
  - NodeSet
  - String
  - Boolean
  - Number

27

XPath : fonctions nodeSet

- *number* **last**(): Retourne la position du dernier noeud
- *number* **position**(): Retourne la position du noeud courant
  - produit[(position() mod 2) = 0]
- *number* **count**(*node-set*): Retourne le nombre de noeuds dans l'ensemble de noeuds
- *node-set* **id**(*object*): Sélectionne les éléments par leur identifiant unique
  - produit[prix > id('CD')/prix]
- *string* **name|local-name|namespace-uri**(*node-set*): Retourne "les" noms des noeuds

28

XPath : fonctions String

- *string* **string**(*object*?):
  - Converti un objet en chaîne de caractères
- *boolean* **concat**(*str1*, *str2*):
  - Retourne la concaténation de str1 et de str2
- *boolean* **start-with**(*str*, *sub*):
  - Retourne vrai si str commence par sub
- *boolean* **contains**(*str*, *sub*):
  - Retourne vrai si str contient sub
- *string* **substring-before|substring-after**(*str1*, *str2*):
  - Retourne la sous-chaîne de str1 qui précède|suit la première occurrence de str2

29

XPath : fonctions String(2)

- *string* **substring**(*str*, *number1*, *number2*?):
  - Retourne la sous-chaîne commençant à l'indice number1 et de longueur number2
- *number* **string-length**(*str*?):
  - Retourne la longueur de str
- *string* **normalize-space**(*str*?):
  - Retourne la chaîne str avec une normalisation des espaces
- *string* **translate**(*str1*, *str2*, *str3*):
  - Retourne str1 avec les occurrences de str2 remplacées par str3

30

XPath : fonctions boolean

- *boolean* **boolean**(*object*):
  - Converti un objet en boolean
- *boolean* **not**(*boolean*):
  - Retourne vrai si l'argument est faux et le vice-versa
- *boolean* **true**():
  - Retourne vrai
- *boolean* **false**():
  - Retourne faux
- *boolean* **lang**(*str*):
  - Retourne vrai ou faux en fonction du langage du contexte du noeud, spécifié par xml:lang

31

XPath : fonctions Number

- *number* **number**(*object*):
  - Converti un objet en nombre
- *number* **sum**(*node-set*):
  - Retourne la somme des valeurs numériques des noeuds passés en arguments
- *number* **floor**(*number*):
  - Retourne le plus grand entier inférieur à number
- *number* **ceiling**(*number*):
  - Retourne le plus petit entier inférieur à number
- *number* **round**(*number*):
  - Retourne l'entier arrondi le plus proche

32

XPath : conclusion

- XPath est très puissant, un vrai langage de requêtes, mais du coup:
- difficile à maîtriser, car concis
- très difficile à implémenter efficacement (prédicats)
- **La force**:
- utilisé de plus en plus par les outils
- comme les expressions régulières: permet d'exprimer la logique d'une action de façon portable entre langages
- **Danger**: pour XPath, un "noeud" veut dire aussi bien attribut qu'élément

33

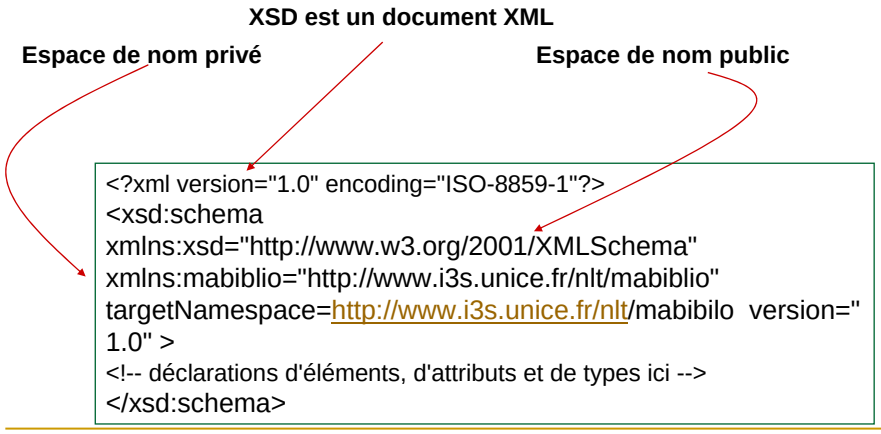
XML Schema : introduction

- Objectifs :
  - étendre le système des DTDs en introduisant des types plus complexes,
  - séparation des types et de la structure,
  - une approche objet (héritage),
  - une syntaxe XML
  - support de la notion d'espaces de noms
- La version 1.0 est propose par le consortium W3C
- en 2001
- La solution proposée est complexe.
- Tous les analyseurs ne traitent pas les schemas

34

XML Schema : introduction (2)

- Format type d'un schéma



35

XML Schema : introduction (3)

- **<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema … /xsd>** Cela signifie que dans le document, tous les éléments commençant par **xsd** sont référencés à cette URL de type **PUBLIC**, où le schéma est... public.
- Un schéma est en effet un document XML, et on trouve dans son élément racine l'attribut. Si on a déposé un schéma à l'adresse xmlns="http://www.i3s.unice.fr/nlt/mabiblio", on peut l'appeler par :

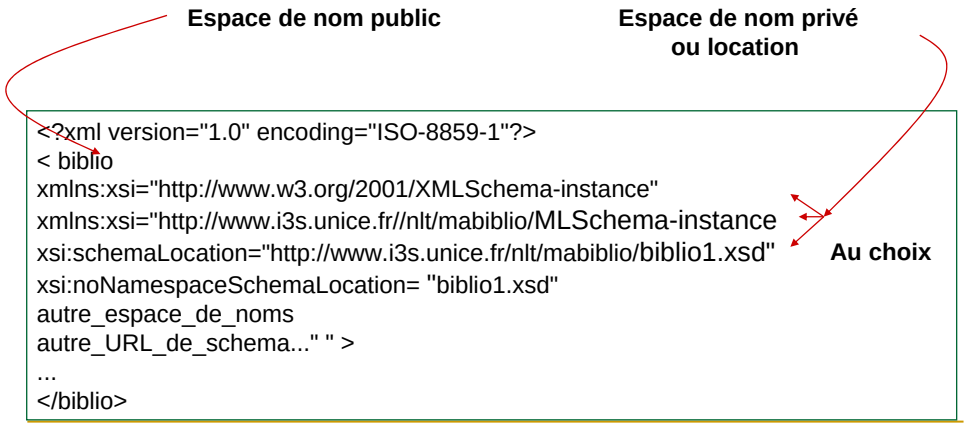
  <xsd:mabiblio xmlns="http://www.i3s.unice.fr/nlt/mabibilo **"> ou**

  targetNamespace=http://www.i3s.unice.fr/nlt/mabibilo

36

XML Schema : introduction (4)

- Lier un fichier xml à un schéma



37

XML Schema : introduction (5)

- Dans le cas d'une référence locale (correspondant à une DTD de type **SYSTEM)** on fait référence au schéma dans le document XML en utilisant l'attribut **noNamespaceSchemaLocation**, par
  - **<biblio xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="biblio10.xsd"> ...>**
  - **<biblio xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"** xmlns:mabiblio="http://www.i3s.unice.fr//nlt/mabiblio xsi:**SchemaLocation="**http://www.i3s.unice.fr//nlt/mabiblio**">**.

38

XML Schema : déclaration des éléments

- Deux manières :
  - Définir un élément à partir d'un type :
    <xsd:element name="prix" type="typePrix" />
  - ou directement :
    <xsl:element name="prix">
    ... definition du type ...
    </xsd:element>
- Les attributs font partie du type.

XML Schema : déclaration des éléments (2)

- Définition depuis d'un type

```
<xsd:element
id="xsd:ID"
abstract = "xsd:boolean"
default="value"
fixed="xsd:string"
minOccurs="xsd:nonNegativeInteger"
maxOccurs="xsd:nonNegativeInteger|unbounded"
name="xsd:NCName"
ref="xsd:QName"
type="xsd:QName"
...
>
(xsd:annotation)
(xsd:simpleType|xsd:complexType)
</xsd:element>
```

XML Schema : déclaration des attributs

- Balise de définition d'un attribut

```
<xsd:attribute
id="xsd:ID"
default="value"
fixed="xsd:string"
name="xsd:NCName"
ref="xsd:QName"
type="xsd:QName"
use="optional|prohibited|required"
...
>
(xsd:annotation)
(xsd:simpleType)
</xsd:attributt>
```

Remarque :
La valeur par défaut doit être conforme au type déclaré

L'exemple :

```
<xsd:attribute
name="maj"
type="xsd:date"
use="optional"
default="-43" />
```

provoquera une erreur de validation du schéma

41

XML Schema : déclaration des attributs (2)

DTD	use	default	Commentaire
#REQUIRED	required	-	
"blabla" #REQUIRED	required	blabla	
#IMPLIED	optional	-	
"blabla" #IMPLIED	optional	blabla	
-	prohibited	-	Cet attribut ne doit pas apparaître

42

XML Schema :hiérarchie de types

xsd:anyType
| xsd:anySimpleType
| | xsd:string
| | | xsd:normalizedString
| | | | xsd:token
| | | | | xsd:name
| | | | | | xsd:NCName
| | | | | | | xsd:ID
| | | | | | | xsd:IDREF
| | | | | | | | xsd:IDREFS
| | | | | | | xsd:ENTITY
| | | | | | | | xsd:ENTITIES
| | | | | xsd:NMTOKEN
| | | | | | xsd:NMTOKENS
| | | | | xsd:langage
| | | xsd:decimal
| | | xsd:integer
| | | | xsd:nonPositiveInteger
| | | | | xsd:NegativeInteger
| | | | xsd:nonNegativeInteger
| | | | | xsd:PositiveInteger
| | | | ... les longs ...
| | ... les durees ...

- Les types de base

a/ Les chaînes
- xsl:string
- xsd:normalizedString
- xsd:token

b/ Les données binaires :
- xsd:base64Binary
- xsd:hexBinary

c/ Les décimaux :
- xsd:decimal
- xsd:float et xsd:double avec les infinis

43

XML Schema :les types de base (2)

d/ Les entiers :
- xsd:integer
- xsd:positiveInteger
- xsd:negativeInteger
- xsd:nonNegativeInteger
- xsd:nonPositiveInteger
-xsd:int
-xsd:unsignedInt (32 bits)
-xsd:long
-xsd:unsignedLong (64 bits)

xsd:short
xsd:unsignedShort (16 bits)
xsd:byte
xsd:unsignedByte (8 bits)
-xsd:boolean (true, false, 0, 1)

e/ Date et durée :
- xsd:time (HH :MM :SS.sss)
- xsd:date (YYYY-MM-DD)
- xsd:dateTime (dateTtime)

44

XML Schema : les types de base (3)

f/ Les noms :
- xsd:Name nom simple
- xsd:QName nom qualifié
- xsd:anyURI

g/ Les clefs et les références :
- xsd:ID
- xsd:IDREF
- xsd:IDREFS

h/ Les données de DTD :
- xsd:ENTITY
- xsd:ENTITIES
- xsd:NOTATION
- xsd:NMTOKEN
- xsd:NMTOKENS

XML Schema : les types simple

- Type simple (#PCDATA) :
  - provient de la restriction ou de l'extension d'un type de base :

```
<xsd:simpleType name="poids">
<xsd:restriction base="xsd:int">
<xsd:minInclusive value='100'/>
</xsd:restriction>
</xsd:simpleType>
```

  - Un type simple peut restreindre un autre type simple :

```
<xsd:simpleType name="poidsMoyen">
<xsd:restriction base="poids">
<xsd:minInclusive value='200'/>
</xsd:restriction>
</xsd:simpleType>
```

XML Schema : les types simples - facettes

- Les facettes sont des restrictions de types simples :

```
<xsd:minInclusive value='valeur'/>
<xsd:minExclusive value='valeur'/>
<xsd:maxInclusive value='valeur'/>
<xsd:maxExclusive value='valeur'/>
```

- Une énumération est possible

```
<xsd:restriction base="xsd:string">
<xsd:enumeration value='rouge'/>
<xsd:enumeration value='noir'/>
<xsd:enumeration value='bleu'/>
</xsd:restriction>
```

- Exemple

```
<xsd:attribute name="jour" type="typeJ
    ourSemaine" use="required" />
<xsd:simpleType name="jourSemaine">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="lundi" />
<xsd:enumeration value="mardi" />
<xsd:enumeration value="mercredi" />
<xsd:enumeration value="jeudi" />
<xsd:enumeration value="vendredi" />
<xsd:enumeration value="samedi" />
<xsd:enumeration value="dimanche" />
</xsd:restriction>
</xsd:simpleType>
```

47

XML Schema : les types simples - facettes (2)

- on peut agir sur la forme :

```
<xsd:restriction base="xsd:string">
<xsd:pattern value='[dD][iI}[lL].*'/>
<xsd:maxLength value='20'/>
<xsd:minLength value='5'/>
</xsd:restriction>
```

- La facette xsd:length est aussi disponible.

```
<xsd:restriction base="xsd:float">
<xsd:totalDigits value='8'/>
<xsd:fractionDigits value='5'/>
</xsd:restriction>
```

48

XML Schema : les types simples - facettes (3)

- finalement, nous pouvons contrôler les blancs :

```
<xsd:restriction base="xsd:string">
<xsd:whiteSpace value='collapse'/>
<xsd:minLength value='5'/>
</xsd:restriction>
```

- « replace » change les tabulations et les retours chariot en blancs.
- « preserve » n'altère pas les blancs.

49

XML Schema : les types simples - extensions

- Liste de valeurs :

```
<xsd:simpleType name="listeAges">
<xsd:list itemType="age"/>
</xsd:simpleType>
```

- Union de valeurs :

```
<xsd:simpleType name="ageInconnu">
<xsd:union memberTypes="age ">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:enumeration value='inconnu'/>
</xsd:restriction>
</xsd:simpleType>
</xsd:union>
</xsd:simpleType>
```

50

XML Schema : les types complexes

- Un élément de type "complexe " peut contenir de sous-élément.
- On peut alors déclarer :
  - des séquences d'éléments, d'attributs ou les deux
  - des types de choix ou
  - des contraintes d'occurences

51

XML Schema : les types complexes (2)

- Élément Sequence :
  - permettant d'indiquer un contenu ordonné et éventuellement répéte.
  - On utilise pour ce faire l'élément **xsd:sequence**, qui reproduit des expressions du langage DTD

- Syntaxe :

```
<xsd:sequence
id="xsd:ID"
minOccurs="xsd:nonNegativeInteger"      1
maxOccurs="xsd:nonNegativeInteger|unbounded" 1 >
(xsd:annotation?)
(xsd:element|xsd:group|xsd:choice|xsd:sequence|xsd:any)*
</xsd:sequence>
```

52

XML Schema : les types complexes (3)

- Exemple de séquences
  - **<xsd:complexType>**
    <xsd:sequence>
    <xsd:element name="nom**" type="**xsd:string**" />**
    <xsd:element name="prénom**" type="**xsd:string**" />**
    <xsd:element name="dateDeNaissance**" type="**xsd:date**" />**
    <xsd:element name="adresse**" type="**xsd:string**" />**
    <xsd:element name="adresseElectronique**" type="**xsd:string**" />**
    <xsd:element name="téléphone**" type="**numéroDeTéléphone**" />**
    </xsd:sequence>
    </xsd:complexType>
  - **une déclaration d'élément équivalent pourrait être faite avec un typage faible, dans une DTD, où apparaîtrait**

(nom, prénom, dateDeNaissance, adresse, adresseElectronique, téléphone)

53

XML Schema : les types complexes (4)

- Élément « choice »:
  - L'élément xsd:choice permet d'exprimer un contenu alternatif et éventuellement répéte.
  - Syntaxe :

    <xsd:choice
    id="xsd:ID" ?
    minOccurs="xsd:nonNegativeInteger" 1
    maxOccurs="xsd:nonNegativeInteger|unbounded" 1 >
    (xsd:annotation?)
    (xsd:element|xsd:group|xsd:choice|xsd:sequence|xsd:any)*
    </xsd:choice>

54

XML Schema : les types complexes (5)

- Exemple de l'élément « choice »

```
<xsd:complexType name="typePersonne">
   <sequence>
     <xsd:element name="nom" type="xsd:string" />
     <xsd:element name="prénom" type="xsd:string" />
     <xsd:element name="dateDeNaissance" type="xsd:date" />
     <xsd:choice>
       <xsd:element name="adresse" type="xsd:string" />
       <xsd:element name="adresseElectronique" type="xsd:string" />
     </xsd:choice>
     <xsd:element name="téléphone" type="numéroDeTéléphone" />
   </sequence>
</xsd:complexType>
```

55

XML Schema : les types complexes (6)

- Élément « all » :
  - L'élément xsd:all permet d'indiquer une suite d'éléments sans préciser l'ordre d'apparition.
  - Cet élément **xsd:all** doit être un enfant direct de l'élément **xsd:complexType**
  - Cet élément est une nouveauté par rapport aux DTD

- Syntaxe

```
<xsd:all
id="xsd:ID" ?
minOccurs="0|1" 1
maxOccurs="1"  >
(xsd:annotation?)
(xsd:element*)
</xsd:all>
```

56

XML Schema : les types complexes (7)

- Exemple de « all » :

```
<xsd:complexType>
 <xsd:all>
  <xsd:element name="nom" type="xsd:string" />
  <xsd:element name="prénom" type="xsd:string" />
  <xsd:element name="dateDeNaissance" type="xsd:date" />
  <xsd:element name="adresse" type="xsd:string" />
  <xsd:element name="adresseElectronique" type="xsd:string" />
  <xsd:element name="téléphone" type="numéroDeTéléphone" />
 </xsd:all>
</xsd:complexType>
```

  - **indique que chacun de ces éléments peut apparaître une fois ou pas du tout (équivalent de l'opérateur ? dans une DTD), et que l'ordre des éléments n'a pas d'importance (cela n'a pas d'équivalent dans une DTD)**

57

XML Schema : les types complexes (8)

- Groupage :
  - L'élément xsd:group permet de regrouper un contenu complexe
  - Syntaxe

```
<xsd:group
name="xsd:NCName" ?
ref="xsd:NCName" ?
minOccurs="xsd:nonNegativeInteger" 1
maxOccurs="xsd:nonNegativeInteger|unbounded" 1 >
(xsd:annotation?)
(xsd:all|xsd:choice|xsd:sequence)
</xsd:group>
```

58

XML Schema : les types complexes (9)

- Modèle lible « any »: tout type d'éléments
  - Pour décrire un choix multiple de types
  - ##any : tous les éléments
  - ##other : éléments hors de l'espace de noms cible
  - ##targetNamespace : éléments de l'espace de noms cible
  - ##local : éléments sans espace de noms
  - strict : déclaration et validation,
  - skip : pas de validation,
  - lax : validation si déclaration

- Syntaxe

```
<xsd:any
id="xsd:ID" ?
minOccurs="xsd:nonNegativeInteger" 1
maxOccurs="xsd:nonNegativeInteger|unbounded" 1
namespace="##any | ##other |
xsd:anyURI* ##targetNamespace? ##local?"
processContents="lax|skip|strict" strict >
(xsd:annotation?)
</xsd:any>
```

59

XML Schema : les types complexes (10)

- Type complexe à partir des types simples
  - Par restriction ou extension des types simples
  - Syntaxe

```
<xsd:complexType ... >
    (xsd:annotation?)
    <xsd:simpleContent>
    (xsd:annotation?)
    xsd:restriction ou xsd:extension
    </xsd:simpleContent>
</xsd:complexType>
```

60

XML Schema : les types complexes (11)

- Exemple de restriction

```
<xsd:simpleType name="monEntier">
    <xsd:restriction base="nonNegativeInteger">
      <xsd:maxExclusive value="100" />
    </xsd:restriction>
</xsd:simpleType>
```

61

XML Schema : les types complexes (12)

- Type complexe à partir des types simples
  - Par extension d'un type simple :
  - Syntaxe

```
<xsd:extension
id="xsd:ID" ?
base="xsd:QName"
>
(xsd:annotation?)
... definitions des attributs ...
</xsd:complexType>
```

62

XML Schema : les types complexes (13)

- Exemple : type complexe depuis d'un type simple
  - **<xsd:complexType name="**typePoids**">**
    <xsd:simpleContent>
    <xsd:extension base="xsd:positiveInteger**">**
    <xsd:attribute name="unite**" type="**xsd:string**" />**
    </xsd:extension>
    </xsd:simpleContent>
    </xsd:complexType>
    - On a *dérivé* un type complexe à partir du type simple **positiveInteger**
    - L'élément **<xsd:simpleContent** indique que le nouvel élément ne contient pas de sous-élément

63

XML Schema : les types complexes (14)

- Définition d' type complexe par dérivation d'un autre type complexe
  - On peut denir un contenu complexe en appliquant une extension ou une restriction a un autre type complexe
  - La valeur de l'attribut « mixed » doit être cohérente avec celle définie dans le type complexe

- Syntaxe

<xsd:complexType ... >
(xsd:annotation?)
<xsd:complexContent
mixed="xsd:boolean" >
(xsd:annotation?)
(xsd:restriction|xsd:extension)
</xsd:complexContent>
</xsd:complexType>

64

XML Schema : les types complexes (15)

- Extension d'un type complexe

```
<xsd:extension
id="xsd:ID" ?
base="xsd:QName"
>
(xsd:annotation?)
(xsd:group|xsd:all|xsd:choice|xsd:sequence)?
... definitions des attributs ...
</xsd:complexType>
```

- Restriction d'un type complexe

```
<xsd:restriction
id="xsd:ID" ?
base="xsd:QName"
>
(xsd:annotation?)
(xsd:group|xsd:all|xsd:choice|xsd:sequence)?
... definitions des attributs ...
</xsd:complexType>
```


XML Schema : les types complexes (16)

- Indicateurs des occurrences
  - Dans une DTD, un indicateur d'occurence ne peut prendre que les valeurs 0, 1 ou l'infini.
  - XML Schema permet de déclarer directement une telle occurence, car tout *nombre entier non négatif* peut être utilisé.

Dans une `DTD`	Valeur de `minOccurs`	Valeur de `maxOccurs`
`*`	0	`Unbounded (illimté)`
`+`	1 (pas nécessaire, valeur par défaut)	`unbounded`
`?`	0	1 (pas nécessaire, valeur par défaut)
rien	1 (pas nécessaire, valeur par défaut)	1 (pas nécessaire, valeur par défaut)

XML Schema : Autres fonctionnalités

- Inclusion des schémas

  <xsd:include schemaLocation="http://www.i3s.unice.fr/nlt/schemas/biblio.xsd**" />**

  - **Il s'agit d'une sorte de "copier-coller" du contenu de la bibliographie dans le schéma en cours d'écriture.**
  - **La seule condition est que le targetNameSpace soit le même dans le Schema XML inclus et dans le Schema XML importateur**

67

XML Schema : Autres fonctionnalités(2)

- Documentation (annotation)
  - Par commentaires comme d'autres document xml
  - Par éléments **xsd:documentation** pour la documentation à l'intention des lecteurs humains
  - Par éléments **xsd:appinfo** pour les informations à l'intention de programmes
  - Ces deux éléments doivent être placés dans un élément **xsd:annotation** qui dispose d'attributs optionnels : **xml:lang** et **source**
  - Les élémenst **xsd:annotation** peuvent être ajoutés au début de la plupart des constructions

68

XML Schema : Autres fonctionnalités(3)

- Attribut « null »
  - pour indiquer explicitement qu'un élément est non renseigné plutôt que d'omettre cet élément.
  - XML Schema intègre un mécanisme similaire que celui de SGBDR permettant d'indiquer qu'un élément peut être non renseigné.
    - Exemple :

```
<personne>
    <nom>Jean Dupont</nom>
    <courriel xsi:null></courriel>
</personne>
```

  - **Cet attribut doit toujours être préfixé par xsi.**
  - **L'élément portant cet attribut, peut contenir d'autres attributs, mais pas de sous-élément**