

# Architecture orientée services (SOA)

## Une introduction à XML

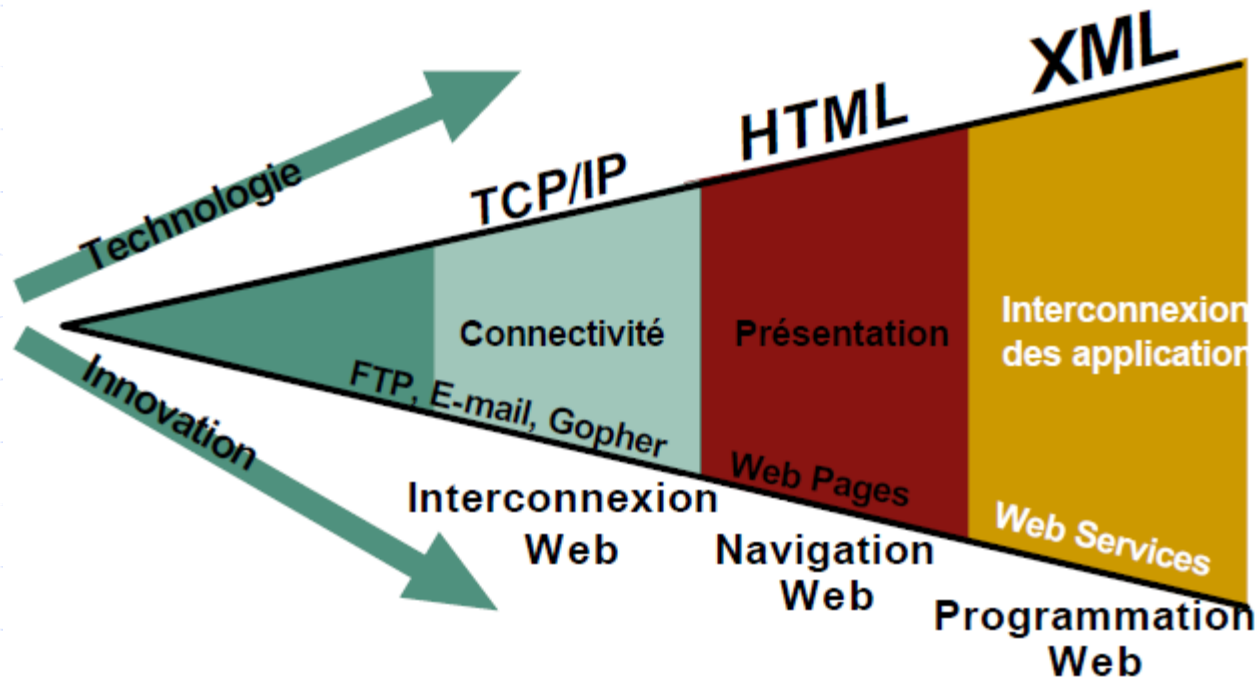
Inès MOUAKHER-ABDELMOULA

3<sup>ème</sup> LFIG

# Plan

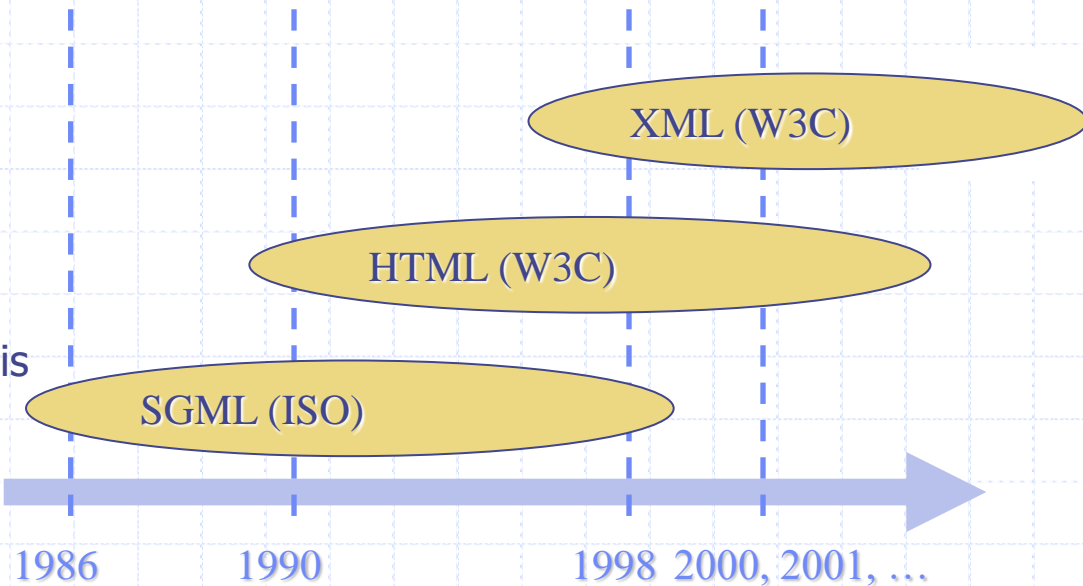
- ◆ Introduction
- ◆ Le langage XML
- ◆ Document Type Definition (DTD )
- ◆ eXtensible Stylesheet Language(XSL)

# Evolution des technologies



# Historique

- **1986 : SGML (*Standard Generalized Markup Language*)**, ISO:8879:1986
  - Trop complexe et coûteux !
- **1990: HTML 1.0 (*HyperText Markup Language*)**
  - Application *pauvre* de SGML
  - Simple, largement utilisé mais non extensible !
- **1998 : XML 1.0 (*eXtensible Markup Language*)**
  - Un SGML allégé !
  - Une recommandation du W3C



# Qu'est-ce que c'est ?

- ◆ XML, eXtensible Markup Language
- ◆ XML 1.0 recommandation du World Wide Web Consortium (W3C) (10 février 1998)
- ◆ Sous-ensemble de SGML
  - Langage à balises
  - 80% des fonctionnalités de SGML, 20% de sa complexité
- ◆ Méta-langage
  - Balises personnalisées
- ◆ Séparer le contenu, la structure et la présentation
- ◆ Indépendant de toutes plate-formes et de tous langages

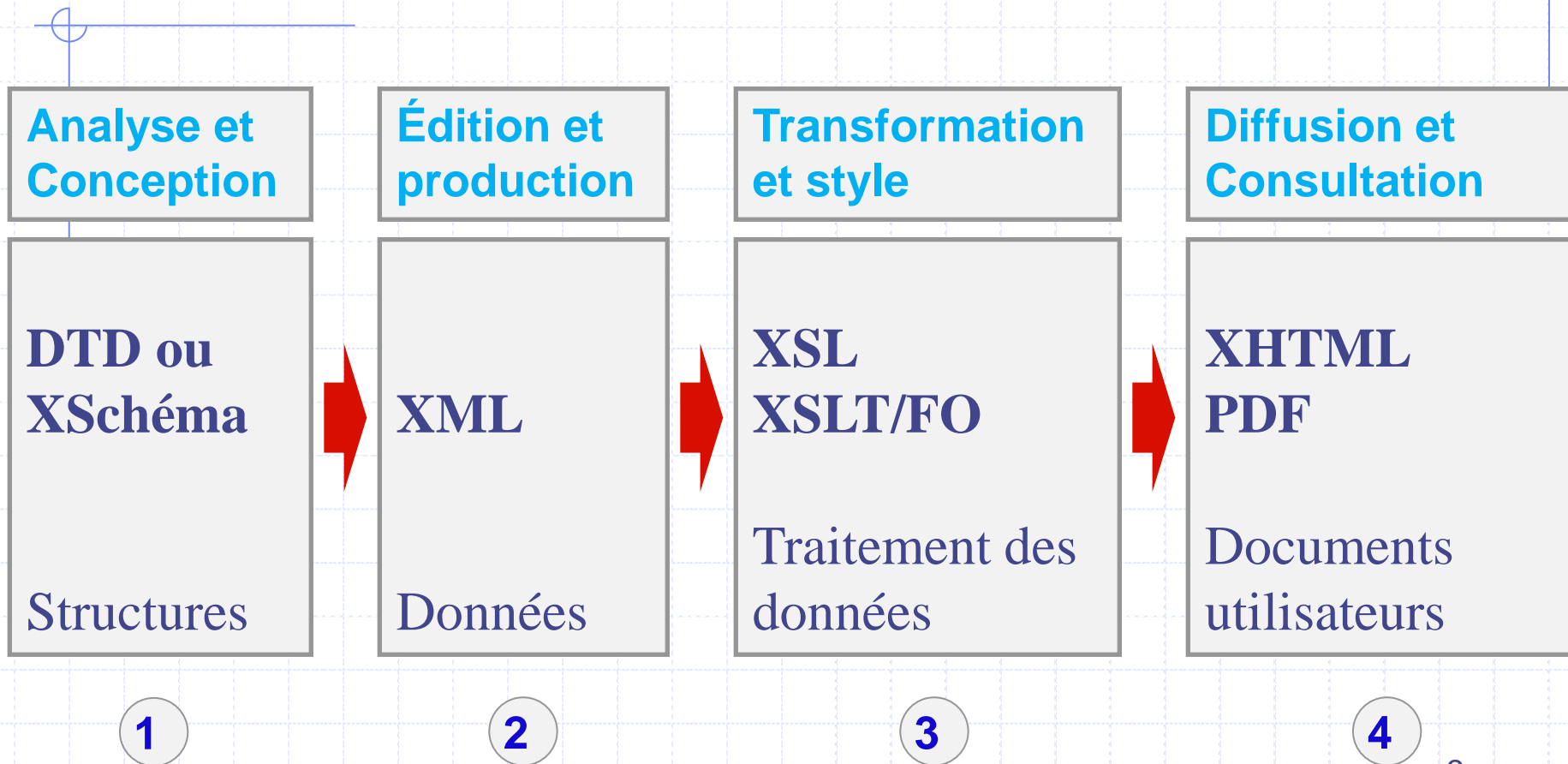
# Principes

1. XML devra pouvoir être utilisé sans difficulté sur Internet
2. XML devra supporter une grande variété d'applications
3. XML devra être compatible avec SGML
4. Il devra être facile d'écrire des programmes traitant les documents XML
5. Le nombre d'options dans XML doit être réduit au minimum
6. Les documents XML devraient être lisible par l'homme
7. La conception de XML devraient être préparée rapidement
8. La conception de XML sera formelle et concise
9. Il devra être facile de créer des documents XML
10. La concision dans le balisage de XML a peu d'importance

# Contenu, Structure et Présentation

- ◆ Document XML : Contenu
  - balises : pas de signification prédéfinie
  - pas de présentation prédéfinie
- ◆ Grammaire de document :
  - DTD : Document Type Definition
  - ou XML-Schéma
- ◆ Présentation et/ou traitement : feuilles de style
  - CSS : Cascading Style Sheets
  - XSL : eXtensible Style Language

# Pour se situer un peu ...





# Le langage XML

- Déclaration
- Racine
- Élément
- Attribut
- Entité
- Règles du développement XML

# Données structurées

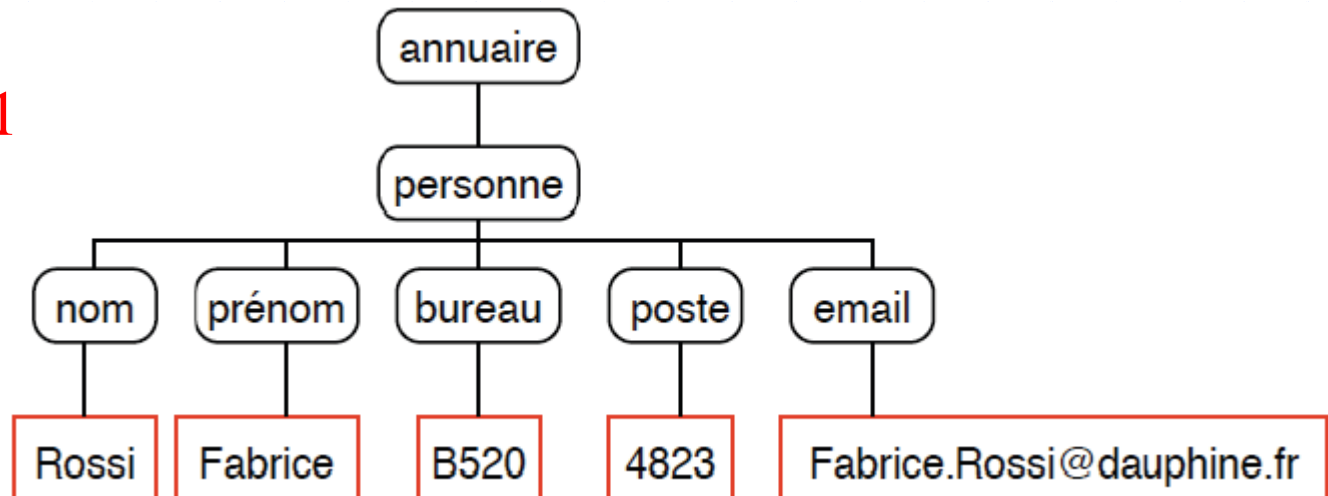
- ◆ XML permet de représenter des données structurées
  - données textuelles
  - organisées :
    - ◆ on manipule un **document** constitué **d'éléments**
    - ◆ un **élément** peut être constitué simplement de texte ou contenir d'autres éléments (ou un mélange des deux)
    - ◆ un élément peut être associé à des informations complémentaires, les **Attributs**
  - la structure est celle d'un arbre :
    - ◆ un document XML = un arbre
    - ◆ un élément = un noeud de l'arbre
- ◆ Le standard XML indique comment traduire l'arbre en un document XML, **pas comment organiser les données**

# Exemple : Annuaire

- ◆ But : stocker l'annuaire de Dauphine (nom, prénom, bureau, numéro de poste, email)
- ◆ Le texte du document : les informations !
- ◆ Organisation : s'arranger pour que les informations restent correctement groupées (ne pas mélanger les données !)

# Exemple : Annuaire

## Solution 1

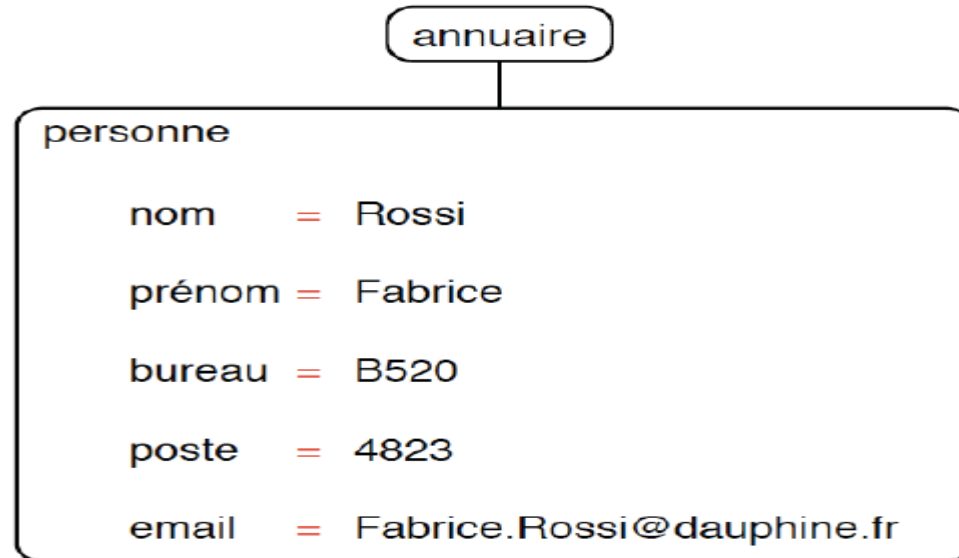


```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <annuaire>
3   <personne>
4     <nom>Rossi</nom>
5     <prenom>Fabrice</prenom>
6     <bureau>B520</bureau>
7     <poste>4823</poste>
8     <email>Fabrice.Rossi@dauphine.fr</email>
9   </personne>
10  <!-- suite de l'annuaire -->
11 </annuaire>
```

annuaire1.xml

# Exemple : Annuaire

## Solution 2



1  
2  
3  
4  
5  
6  
7  
8  
9  
10

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<annuaire>
  <personne nom="Rossi"
            prenom="Fabrice"
            bureau="B520"
            poste="4823"
            email="Fabrice.Rossi@dauphine.fr"/>
  <!-- suite de l'annuaire -->
</annuaire>
```

annuaire2.xml

# Déclaration

## Syntaxe

```
<?xml version="version" [encoding="encodage"] [standalone="yes | no"]?>
```

Cette déclaration (qui est en fait une instruction de traitement) contient des informations pour le processeur. Elle indique que ce document est conforme à la version 1.0 de la norme XML. Elle peut préciser le jeu de caractères utilisés dans le document (encoding) et s'il y a des références externes ou non (standalone).

## Exemple:

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

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

# Déclaration : Encoding

- ◆ les parseurs xml doivent supporter au minimum utf-8 et utf-16.
- ◆ L'encodage recommandé pour les documents XML est UTF-8, garantissant portabilité et multilinguisme.
- ◆ UTF-16 est un autre format permettant l'encodage de documents XML.
- ◆ Il est également possible d'utiliser un encodage spécifique, exemple ISO 8859-1, pour les langues latines

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

- ◆ Voir <http://www.unicode.org>
  - Latin/Arabe : ISO-8859-6,
  - Latin/Lapon/Nordique/Esquimau : ISO-8859-10;
  - Japonais : EUC-JP

# Imbrication des éléments

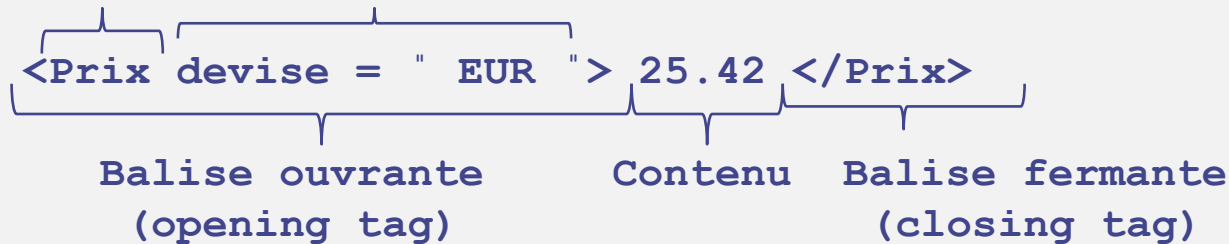
- ◆ Tout document XML doit comporter une racine
- ◆ Chaque élément du document XML peut contenir un ou plusieurs éléments.
- ◆ Chaque document XML est hiérarchisé dans une arborescence.
- ◆ Les balises d'ouverture et de fermeture des éléments fils doivent toujours être comprises entre les balises d'ouverture et de fermeture des parents.



# Élément

- ◆ La base d'un document XML

Non de l'élément Attribut



- ◆ Quand utiliser les attributs?
  - Valeur unique de - type simple (information monovaluée)
- ◆ Quand utiliser les éléments?
  - Valeur de type complexe (énumérations, possède des propriétés)

# Élément

- ◆ Les balises XML peuvent être de deux sortes ; la première contient, entre la balise de début et la balise de fin, des données diverses comme du texte.

```
<Élément>donnée</Élément>
```

- ◆ La seconde est du type vide, c'est-à-dire, des balises ne contenant pas de données à proprement parler

```
<Élément></Élément>
```

```
<Élément/>
```

# Les Balises

- ◆ Un élément constitutif d'un document XML repose sur un nom et un contenu.
- ◆ Le contenu d'un élément est compris entre :
  - Une balise de début (balise d'ouverture).
  - Une balise de fin (balise de fermeture).
- ◆ Balise de fermeture = balise d'ouverture + /
- ◆ `<pays>Tunisie</pays>`
- ◆ Contrairement à HTML, la balise de fermeture est nécessaire dans XML.

# Les noms de balises

- ◆ Les noms peuvent contenir des caractères alphanumériques, de soulignement (\_), de ponctuation (., ,; ) et le trait d'union.
- ◆ Les noms de balises sont sensibles à la casse.  
`<Pays> # <pays>`
- ◆ Les noms doivent commencer par un caractère alphabétique ou par le soulignement (\_).
- ◆ Certains caractères spéciaux sont permis mais peuvent fausser l'interprétation au niveau de certains programmes (> + é à ù ï ç)
- ◆ Les noms ne peuvent pas contenir des espaces
- ◆ Les noms ne peuvent pas commencer par la séquence `xml`
- ◆ Les caractères interdits sont : `? $ ! < &`

# La racine

- ◆ À la racine d'un document XML, il ne peut y avoir qu'un seul et unique élément.
- ◆ Le document XML se termine à la fermeture de la balise de racine.

```
<? xml version = "1.0" ?>  
<messenger>  
<contact pseudo= "XXX" >  
</contact>  
</ messenger >
```

```
<? xml version = "1.0" ?>  
<messenger>  
</ messenger >  
<contact pseudo= "XXX" >  
</contact>
```

# Les attributs

- ◆ Objectif: Associer des informations complémentaires aux éléments.

```
<montant monnaie= "DT" > 24 </montant>
```

- ◆ Les noms des attributs suivent les mêmes règles que celles des noms des éléments.
- ◆ Un élément peut posséder un ou plusieurs attributs dans la balise d'ouverture.
- ◆ Le nom de l'attribut est unique pour un élément donné.
- ◆ La valeur de l'attribut doit obligatoirement être entre " ".
- ◆ Les guillemets (" ") peuvent être remplacés par des apostrophes ( ` ` )

# Entité

- ◆ Appel d'une entité dans un document : **&nom\_entite;**
- ◆ Les caractères réservés de XML sont remplacés par des entités internes. Ces caractères sont les mêmes qu'en HTML: & < > " '. Les entités qui permettent de les représenter sont respectivement *&amp;*; *&lt;*; *&gt;*; *&quot;*; *&apos;*;
- ◆ Tous les caractères peuvent être remplacés par une entité qui donne leur code **&#code\_car;**
- ◆ Exemple: *&#65;* pour le A
- ◆ Déclaration d'une entité: **<!ENTITY deg "&#176;">**
- ◆ exemple : *il fait 25&deg;*C.

# Commentaire

- ◆ Faciliter la lecture du code ainsi que sa maintenance.
- ◆ Ces commentaires seront ignorés lors de la compilation.
- ◆ Pour XML les commentaires s'écrivent de la même manière que dans le langage HTML, ils peuvent inclure n'importe quel type de données sauf le --, ils ne peuvent pas apparaître à l'intérieur des balises.
- ◆ `<!--Commentaire-->`



# Règles de développement d'un document XML

Un document XML est bien formé (l'analyseur XML peut construire son arborescence) si :

- ◆ il contient une déclaration XML ;
- ◆ il contient un ou plusieurs éléments ;
- ◆ il contient un élément racine encapsulant tous les autres éléments et leurs attributs
- ◆ les éléments non vides ont une balise de début et de fin ;
- ◆ les éléments non vides sont correctement imbriqués
- ◆ les éléments vides ont un / à la fin de la balise avant le > ;
- ◆ les noms des balises ouvrantes et fermantes correspondent ;
- ◆ un nom d'attribut apparaît uniquement dans la balise ouvrante et une seule fois dans cette balise ;

# Règles de développement d'un document XML

- ◆ les valeurs des attributs sont entre guillemets ou apostrophes ;
- ◆ la valeur des attributs n'appelle pas d'entités externes directement ou indirectement ;
- ◆ les caractères réservés sont remplacés par des références d'entités (par ex. &lt; pour <) ;
- ◆ toutes les références à des entités doivent commencer par & et finir par ;
- ◆ s'il n'y a pas de DTD, les seules entités utilisées sont celles réservées de XML &amp; ; &lt; ; &gt; ; &apos; ; &quot; ;
- ◆ s'il y a une DTD toutes les entités non réservées utilisées sont déclarées dans la DTD.

# Bibliographie

- ◆ <http://www.w3c.org/XML>
- ◆ <http://www.xml.com>
- ◆ <http://fr.wikipedia.org/wiki/XML>
- ◆ <http://xml.apache.org>
- ◆ <http://www.xmlsoftware.com>
- ◆ <news://comp.text.xml>



# **DTD (DOCUMENT TYPE DEFINITION)**

# Plan

- ◆ *Définition de DTD*
- ◆ *Déclaration d'une DTD*
- ◆ *Déclaration des éléments*
- ◆ *Déclaration d'entités*
- ◆ *Déclaration des attributs*
- ◆ *Exemple de DTD interne*
- ◆ *Insuffisance des DTD*

# Bien formé / Valide

- ◆ Document bien formé
  - Un seul élément racine
  - Balises correctement imbriquées : balise ouvrantes ont une balise fermante associée et il n'y a pas de chevauchement ...
  - Le nom des balises est libre mais contient au moins une lettre en début
  - Les attributs des balises ont obligatoirement une valeur qui doit apparaître entre double ou simple quotes.
- ◆ Document valide
  - Associé à une définition de type de document et qu'il la respecte
    - ◆ Noms des éléments
    - ◆ Type
    - ◆ Répétition et ordre d'apparition

# Validation des documents XML

## Syntaxe de haut niveau (grammaire)

- ◆ précisée par une DTD (*Document Type Definition*):
  - existe depuis la norme XML
  - outils stables
  - limitées : structure simple
  - syntaxe non XML
- ◆ précisée par un schéma :
  - deux grandes technologies (d'autres existent) :
    - ◆ les schémas du W3C (recommandation du 2 Mai 2001)
    - ◆ RELAX NG du consortium OASIS (spécification du 12 Décembre 2001)
  - syntaxe XML
  - très puissants
  - Structures assez complexes

# Définition

- ◆ But : Définir une structure type de document XML.
- ◆ Une grammaire qui décrit la façon de construire les documents XML.
- ◆ Une DTD définit la filiation des éléments :
  - Quelle est la racine du document ?
  - Qui doit/peut avoir quels fils ?
  - Quels sont les éléments (attributs) obligatoires et ceux optionnels ?
  - Combien d'enfants possède un élément ?
  - Quels éléments peuvent contenir du texte ?
  - Quelles sont les valeurs des attributs ?



# Exemple de DTD

```
<!ELEMENT MEMBRE  
    (LOGIN, NOM?, PRENOM?, MEL, TEL+, FAX*, EQUIPE)>  
<!ELEMENT LOGIN EMPTY>  
<!ATTLIST LOGIN id ID #REQUIRED>  
  
<!ELEMENT NOM (#PCDATA)>  
...  
<!ENTITY RDP "Recherche et Développement Produits">  
<!ENTITY eacute "&#233;">  
<!ENTITY chap1 SYSTEM "http://.../chapitre-1.xml">  
...
```

# Déclaration d'une DTD - Interne

## ◆ Syntaxe

```
<!DOCTYPE élément-racine [  
déclaration des éléments  
>
```

## ◆ Exemple

```
<?xml version="1.0" standalone="yes"?>  
<!DOCTYPE parent [  
<!ELEMENT parent (garcon,fille)>  
<!ELEMENT garcon (#PCDATA)>  
<!ELEMENT fille (#PCDATA)>  
>  
<parent>  
<garcon>Loic</garcon>  
<fille>Marine</fille>  
</parent>
```

# Déclaration d'une DTD – Externe

## ◆ Syntaxe

```
<!DOCTYPE élément-racine SYSTEM "nom_du_fichier.dtd">
```

## ◆ Exemple

```
<?xml version="1.0" standalone="no"?>  
<!DOCTYPE parent SYSTEM "parent.dtd">  
<parent>  
<garcon>Loic</garcon>  
<fille>Marine</fille>  
</parent>
```

## ◆ Le fichier de DTD externe "parent.dtd"

```
<!ELEMENT parent (garcon,fille)>  
<!ELEMENT garcon (#PCDATA)>  
<!ELEMENT fille (#PCDATA)>
```

# Déclaration des éléments (1/4)

- ◆ Déclaration de chaque élément qui apparaît à l'intérieur d'un fichier XML.
- ◆ Exemple: `<!ELEMENT nom_element (regle)>`
- ◆ `nom_element` obéit aux règles syntaxiques concernant les noms des éléments XML
- ◆ La règle peut avoir la forme suivante
  - Élément vide `<!ELEMENT soc EMPTY>`
  - Élément libre `<!ELEMENT post-scriptum ANY>`
  - Élément (texte) `<!ELEMENT nom (#PCDATA)>`

# Déclaration des éléments (2/4)

- ◆ La règle peut faire référence à séquence d'éléments fils
  - Séquence ordonnée : (fils1,fils2,fils3)

```
<!ELEMENT date (mois, annee)>
```

```
<date>
<mois>10</mois>
<annee>2006</annee>
</date>
```

```
<date>
<annee>2006</annee>
<mois>10</mois>
</date>
```

```
<date>
<annee>2006</annee>
>
</date>
```

- Séquence non ordonnée : (fils1|fils2|fils3)

```
<!ELEMENT date (mois|annee)>
```

```
<date>
<mois>10</mois>
<annee>2006</annee>
</date>
```

```
<date>
<annee>2006</annee>
<mois>10</mois>
</date>
```

```
<date>
<mois>10</mois>
</date>
```

# Déclaration des éléments (3/4)

- Séquence non ordonnée : (fils1&fils2&fils3)

```
<!ELEMENT date (mois&annee)>
```

```
<date>  
<mois>10</mois>  
<annee>2006</annee>  
</date>
```

```
<date>  
<annee>2006</annee>  
</date>
```

```
<date>  
<annee>2006</annee>  
<mois>10</mois>  
</date>
```

## La déclaration de l'élément date

```
<!ELEMENT date (mois, année)>  
<!ELEMENT mois (#PCDATA)>  
<!ELEMENT annee (#PCDATA)>
```

# Déclaration des éléments (4/4)

- ◆ On peut utiliser une séquence ordonnée et une séquence non ordonnée au niveau d'une même déclaration d'un élément.

```
<!ELEMENT livre((titre, auteur) | description)
```

- ◆ Cette déclaration signifie qu'il est possible de saisir
  - soit la description d'un livre
  - soit son titre suivi de son auteur

# Occurrence des éléments

## ◆ Occurrence

- \* : 0 à n occurrences
- + : au moins 1 occurrences
- ? : 0 ou 1 occurrence

```
<!ELEMENT livre (titre,auteur+)>
```

- ◆ Un livre doit avoir un ou plusieurs auteurs.



# Déclaration d'entités (1/3)

## ◆ Entité paramétrée

- L'entité paramétrée n'apparaît qu'au niveau de la
- DTD, elle est remplacée par sa définition dans la DTD.
- Exemple : La déclaration qui suit

```
<!ENTITY %pcdata "(#PCDATA)">
```

- permet de remplacer la suite de caractères (#PCDATA) au niveau de la DTD par %pcdata, on pourra ainsi déclarer l'élément titre comme suit

```
<ELEMENT titre %pcdata>
```

# Déclaration d'entités (2/3)

## Entité externe

- ◆ Faire une référence à des entités externes à travers la syntaxe suivante

```
<!ENTITY exp SYSTEM "http://www.site.com/data.xml" >
```

- ◆ Permet de copier le contenu XML (dans l'URL spécifiée) dans le document XML courant.

```
<livre> &exp; </livre>
```

# Déclaration d'entités (3/3)

- ◆ Entité « non analysée » (Unparsed)

- Déclarer un contenu non XML à l'intérieur d'un document XML
- Exemple : Intégration d'une image

```
<!ENTITY image1 SYSTEM "http://www.site.com/img.gif" NDATA GIF89a>
```

- Au niveau du fichier XML, on procède comme suit

```
<image src="image1">
```

- Remarque : On a enlevé le & ainsi que le ; qui délimitent généralement une entité à l'intérieur d'un fichier XML

# Déclaration des attributs

```
<!ATTLIST      nom_elmt nom-attribut  
                type_attribut val_defaut >
```

- **nom\_elmt** : la balise à laquelle on précise la liste des attributs
- **nom\_attribut** : le nom de l'attribut qu'on va ajouter
- **type\_attribut** : règle qu'un attribut doit respecter
- **val\_defaut** : valeur par défaut

# Déclaration des attributs

- ◆ Si la valeur par défaut n'est pas spécifiée, on peut préciser les paramètres suivants :

- #FIXED "val" La valeur de l'attribut est fixé et ne doit pas être changé

```
<!ATTLIST date année CDATA #FIXED "2004">
```

- #IMPLIED L'attribut est optionnel

```
<!ATTLIST personne age CDATA #IMPLIED>
```

- #REQUIRED L'attribut doit être spécifié.

```
<!ATTLIST personne nom CDATA #REQUIRED>
```

- "val" La valeur par défaut.

```
<!ATTLIST rectangle largeur CDATA "0">
```

# Déclaration des attributs

- ◆ Types possibles pour les attributs :

- Chaîne de caractère littéral

```
<!ATTLIST soc name CDATA #IMPLIED>
```

- ID identifiant dans le document

```
<!ATTLIST soc idname ID #REQUIRED>
```

- IDREF, IDREFS renvoi(s) vers ID à l'intérieur du doc

```
<!ATTLIST soc banque IDREF #IMPLIED>
```

- ENUMERE l'ensemble des valeurs possibles de l'attribut est défini.

```
<!ATTLIST personne fonction (ing|com|tech|admin) "ing">
```

# Déclaration des attributs

- ◆ La notation ID permet d'assurer qu'un attribut a une valeur unique dans tout le document XML.
- ◆ Les notations IDREF et IDREFS permettent de relier les différentes sections d'un document.

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE personnels SYSTEM "LePersonnel.dtd">
<personnels>
<employe ide="a10" directeur="d50" collegues="a40 a60"/>
<employe ide="a20" directeur="d50" collegues="a50"/>
<employe ide="a30" directeur="d20"/>
<employe ide="a40" directeur="d20" collegues="a10 a60"/>
<employe ide="a50" directeur="d20" collegues="a20"/>
<employe ide="a60" directeur="d50" collegues="a10 a40"/>
<employe ide="d20" collegues="d50"/>
<employe ide="d50" collegues="d20"/>
```

# Déclaration des attributs

- ◆ La DTD associée à l'exemple précédent est

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT personnels (employe+)>
<!ELEMENT employe EMPTY>
<!ATTLIST employe ide ID #REQUIRED>
<!ATTLIST employe directeur IDREF #IMPLIED>
<!ATTLIST employe IDREFS #IMPLIED>
```



# Déclaration des attributs

- ◆ ENTITY, ENTITIES référence à une ou plusieurs entités externe non XML

```
<!NOTATION gif SYSTEM "C:\Program\ACDSee32.exe">  
<!ENTITY LigneBleu SYSTEM "logo-auf.gif" NDATA gif>  
<!ELEMENT separateur EMPTY>  
<!ATTLIST separateur img ENTITY #REQUIRED>
```

```
<!ENTITY nom SYSTEM "URI" NDATA type_notation>
```

- ◆ Les entités externes non analysées permettent de déclarer un contenu non XML dans le document XML (fichiers binaires images, sons...).
- ◆ Le mot clé NDATA (Notation DATA) précise le type d'entité non analysée que le processeur XML doit traiter.
- Par exemple, la déclaration d'entité

```
<!ENTITY vacances SYSTEM "images/plage.gif" NDATA GIF89a>
```

- attribue un alias (vacances) à une entité (images/plage.gif) de type GIF89a sauvegardée sur le disque (SYSTEM).

# Exemple

```
<?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>
```

# Exercice (Catalogue de films)

- ◆ On se propose de définir un format XML de stockage d'un catalogue de films sur DVD.
- ◆ Le catalogue comprend un ensemble de fiches de films
- ◆ Chaque fiche comprend:
  - Un numéro unique
  - Le titre du film
  - Un ou plusieurs réalisateurs
  - Un ou plusieurs éditeurs
  - Les acteurs principaux
  - Le genre du film (comédie, horreur, action...) (en option)
  - Un commentaire optionnel qui présente brièvement l'histoire du film
  - Un lien éventuel vers le site du film

# Exercice (Catalogue de films)

```
<!--premier niveau: catalogue -->
  <!ELEMENT cataloguedvd (fiche)*>
<!--deuxième niveau: fiche -->
  <!ELEMENT fiche (titre, technique, commentaire?, internet?)>
  <!ATTLIST fiche genre (Horreur|Action|comédie|inconnu) #IMPLIED>
  <!ATTLIST fiche numero ID #REQUIRED>
<!--troisième niveau: sous éléments de 'fiche' -->
  <!ELEMENT titre (#PCDATA)>
  <!ELEMENT commentaire (#PCDATA)>
  <!ELEMENT internet (#PCDATA)>
  <!ELEMENT technique (realisateur+, editeur+, acteur*)>
  <!--quatrième niveau: sous éléments de 'technique' -->
  <!ELEMENT realisateur (#PCDATA)>
  <!ELEMENT editeur (#PCDATA)>
  <!ELEMENT acteur (#PCDATA)>
```

# Insuffisance des DTD

- ◆ Pas de types de données à part du texte (#PCDATA)
- ◆ Expression de cardinalités limitée ('?', '\*' et '+')
- ◆ Syntaxe spécifique (pas XML)
  - difficile à interpréter
  - difficile à traduire en schéma objets
- ◆ Propositions de compléments
  - XML-schema du W3C



# **PRÉSENTATION ET/OU TRAITEMENT**

# Afficher des documents XML

- ◆ Un document XML ne fournit pas d'information sur sa présentation
- ◆ Affichage personnalisé
- ➔ Feuilles de style :
  - *Casading Style Sheets* (CSS 1 et 2)
    - ◆ plutôt statique, pas de transformation et très orienté HTML/Navigateur.
  - *Extensible Style Language* (XSL)
    - ◆ entièrement dynamique et indépendant
- ◆ Transformation de documents XML

# Cascading Style Sheets

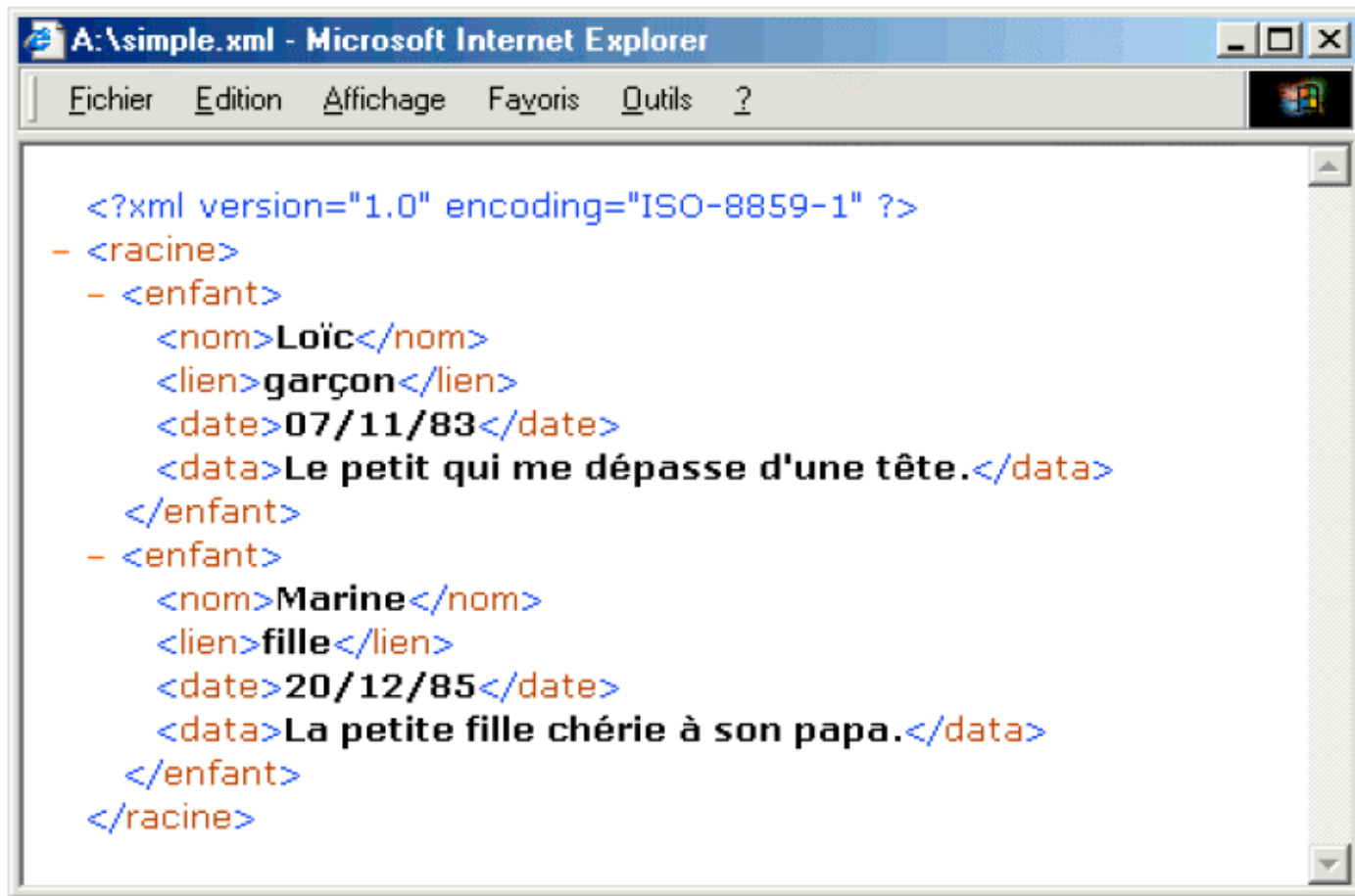
- ◆ Pour afficher les balises XML, on peut faire appel aux bonnes vieilles feuilles de style (CSS)
- ◆ Syntaxe non XML
- ◆ Mise en forme pure
  - Pas de modification de structure
  - Pas de modification de contenu
- ◆ A chaque balise "inventée" dans le fichier XML, on va définir un élément de style que le navigateur pourra alors afficher



# Exemple : document XML

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

# Exemple : Affiché dans le navigateur

A screenshot of a Microsoft Internet Explorer browser window. The title bar reads "A:\simple.xml - Microsoft Internet Explorer". The menu bar includes "Fichier", "Edition", "Affichage", "Favoris", "Outils", and "?". The main content area displays XML code with syntax highlighting. The code defines two children: a boy named Loïc born on 07/11/83, and a girl named Marine born on 20/12/85. The text content for each child is displayed in bold black font.

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

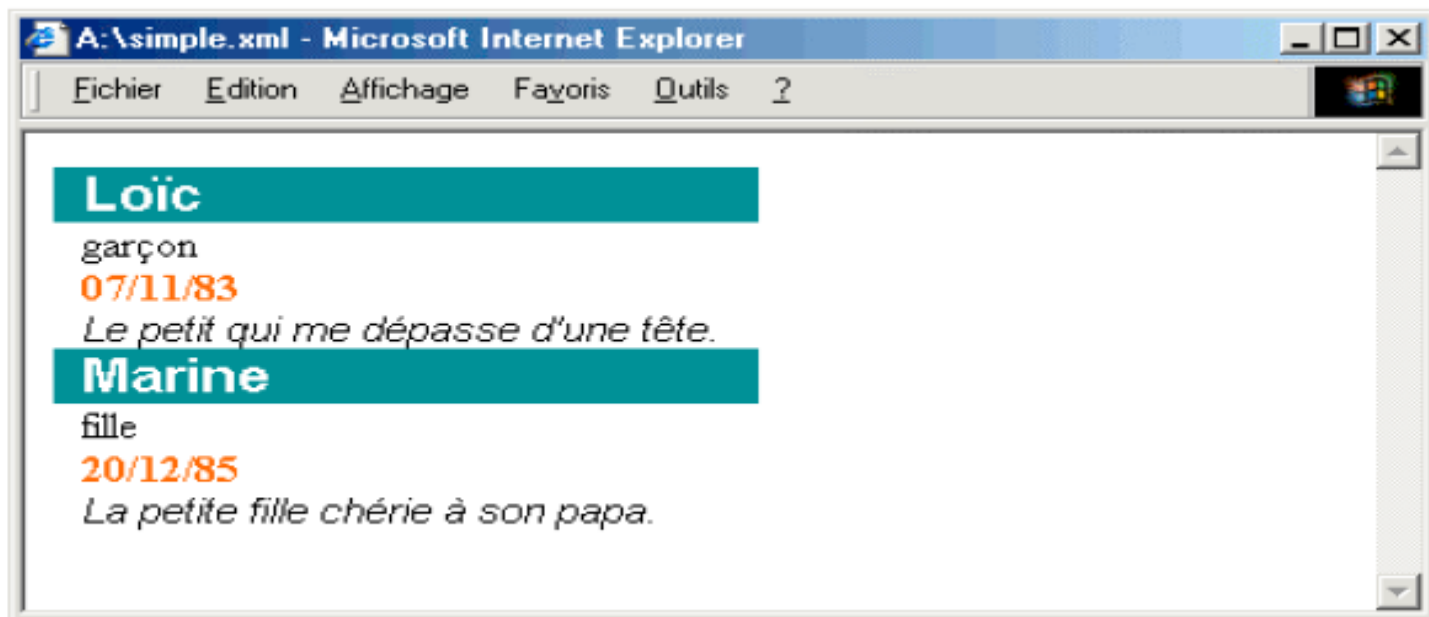
# Exemple : fichier .css

```
<style type="text/css">
racine , enfant {}
nom {
display: block;
width: 250px;
font-size: 16pt ;
font-family: arial ;
font-weight: bold;
background-color: teal;
color: white;
padding-left: 10px;
}
lien {
display: block;
font-size: 12pt;
padding-left: 10px;
}
```

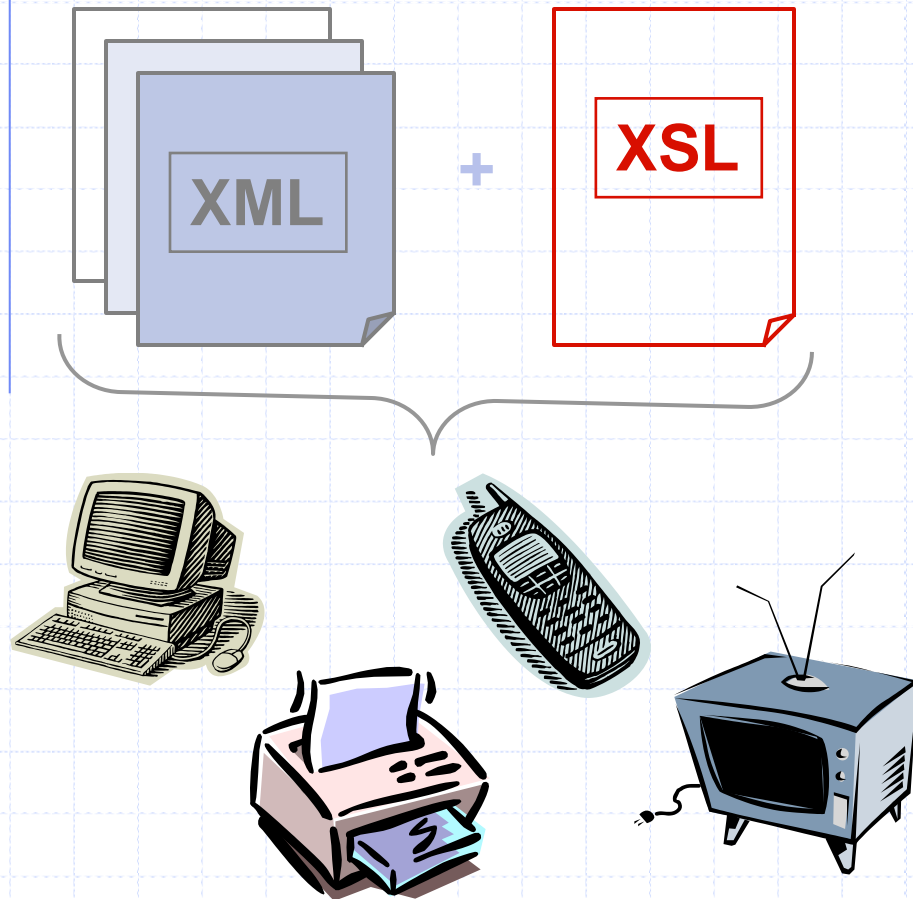
```
date {
display: block;
font-size: 12pt;
color: red ;
font-weight: bold;
padding-left: 10px;
}
data {
display: block;
font-size: 11pt ;
font-style: italic;
font-family: arial ;
padding-left: 10px;
}
</style>
```

# Exemple : Affiché dans le navigateur

- ◆ On ajoute un lien vers le fichier css dans le fichier xml : `<?xml-stylesheet href="css.css" type="text/css"?>`
- ◆ On obtient :



# eXtensible Stylesheet Language



- ◆ Décrit la manière dont les documents XML seront affichés, imprimés ou ... prononcés
- ◆ Indépendant du média de sortie

# eXtensible Stylesheet Language

- ◆ XSL (eXtensible Stylesheet Language) est le langage de description de feuilles de style du W3C associé à XML.
- ◆ Une feuille de style XSL est un fichier qui décrit comment doivent être présentés (c'est-à-dire affichés, imprimés) les documents XML.
- ◆ La spécification est divisée en trois parties :
  - XSLT, le langage de transformation
  - XPath, le langage de navigation dans un document XML
  - XSL-FO, le vocabulaire XML de mise en forme

# Afficher le XML avec XSL

- ◆ Le XSL ne permet pas uniquement l'affichage de XML. Il permet aussi :
  - de sélectionner une partie des éléments XML.
  - de trier des éléments XML.
  - de filtrer des éléments XML en fonction de certains critères.
  - de choisir des éléments.
  - de retenir des éléments par des tests conditionnels.



# XML SCHÉMA



# Plan

- ◆ Introduction
- ◆ En-tête
- ◆ Référence à un Schéma XML
- ◆ Déclarations d'éléments
- ◆ Les attributs
- ◆ Groupage d'éléments

# Introduction

- ◆ 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
  - utilise un espace de nom xs: (ou xsd:)
- ◆ Présente de nombreux avantages
  - types de données personnalisés
- ◆ La notion d'héritage. Les éléments peuvent hériter du contenu et des attributs d'un autre élément..
- ◆ Le support des espaces de nom.

# En-tête

- ◆ Un document Schema XML est défini dans un fichier dont l'extension est \*.xsd (monSchema.xsd)
- ◆ Comme tout document XML, un Schema XML commence par un prologue, et a un élément racine.
- ◆ La balise <schema> est la balise racine de tous documents Schema XML
- ◆ Le corps de la balise <schema> décrit le contenu de la grammaire

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<xsd:schema  
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">  
<!-- déclarations d'éléments, d'attributs et de types ici -->  
</xsd:schema>
```

# En-tête

- ◆ La configuration du document Schema XML est effectuée par l'intermédiaire des attributs de la balise `<schema>`

```
<schema  
id=ID  
attributeFormDefault=qualified|unqualified  
elementFormDefault=qualified|unqualified  
blockDefault=(#all|list of (extension|restriction|substitution))  
finalDefault=(#all|list of (extension|restriction|list|union))  
targetNamespace=anyURI  
version=token  
xmlns=anyURI  
any attributes>
```

# En-tête : Exemple

L'espace de nommage des éléments définis par la norme W3C (ex : schemaLocation)

Précise que les éléments définis dans le Schema XML sont issus dans un autre espace de nommage

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://mbaron.developpez.com/InfoPersoSchema"
  xmlns:as="http://mbaron.developpez.com/InfoPersoSchema"
  elementFormDefault="qualified"
  attributeFormDefault="qualified">
  ... // Contenu du Schema XML
</schema>
```

Précise que pour cet espace de nommage les éléments sont préfixés par as

"qualified" indicates that elements from the target namespace must be qualified with the namespace prefix

# Référencer un Schema XML

- ◆ Un document XML décrit par un XSD est appelé instance document
- ◆ Pour référencer un document XML à un Schema XML plusieurs informations sont à renseigner

L'espace de nommage des éléments définis par la norme W3C

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:person
xmlns:tns="http://mbaron.developpez.com/InfoPersoSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
tns:schemaLocation="http://mbaron.developpez.com/InfoPersoSchema/InfoPersoSchema.xsd ">
...
</tns:person>
```

L'espace de nommage par défaut où tous les éléments seront préfixés par tns

# Référencer un Schema

- ◆ XSD file  
(<http://mbaron.developpez.com/InfoPersoSchema/InfoPersoSchema.xsd> )

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://mbaron.developpez.com/InfoPersoSchema/"
  xmlns="http://mbaron.developpez.com/InfoPersoSchema/"
  elementFormDefault="qualified">
....
</xs:schema>
```

# Déclarations d'éléments

- ◆ La base d'un schéma XML: l'élément
- ◆ Un élément dans un Schema XML se déclare avec la balise <element>
- ◆ Un élément est typé, il doit donc respecter une structure de données
- ◆ Deux types de données sont à distinguer : **types simples** et **types complexes**
- ◆ Type Simple si sa valeur a un type prédéfini en XMLSCHEMA (string, int, decimal, double...) ou une extension de ces types
- ◆ Complexe s'il contient des sous éléments ou s'il comporte un attribut



# Éléments de type simple

- ◆ Un élément de type simple est un élément qui ne contient pas d'attribut ni de sous éléments
- ◆ Un élément se base sur un type simple prédéfini par la norme ou sur un type simple dérivé (voir dans Restrictions)
- ◆ Syntaxe d'un élément de type simple

```
<xs:element name="..." type="..."/>
```

- ◆ Exemples

```
<xs:element name="firstName" type="xs:string" />
```

Du côté XML, la balise `firstName` contient un corps avec une valeur de type chaîne de caractères

```
<firstName> Mickael </firstName>
```

# Types simples

- ◆ **string**

Chaine de Caractères

- ◆ **integer**

-126789 – >126789

- ◆ **positiveInteger**

1 -> 126789

- ◆ **negativeInteger**

-126789 -> -1

- ◆ **Boolean**

true, false 1, 0

- ◆ **date**

1999-05-31

- ◆ **ID**

"A212 "

- ◆ **dateTime**

◆ 1999-05-31T13:20:00.000-05:00

◆ Et beaucoup d'autres

**Short, long, floatA....**

# Types simples – Restrictions (1/4)

- ◆ Les restrictions sur les types simples (Facets) permettent de dériver de nouveaux types à partir de types existants
- ◆ La création de nouveaux types simples est réalisée au travers de la balise `<simpleType>`

```
<simpleType name="newType" >  
  ...  
</simpleType>
```

- ◆ La restriction de type permet de définir des contraintes sur le nouveau type à créer
- ◆ La restriction est exprimée à partir de la balise `<restriction>`

```
<simpleType name="newType" >  
  <restriction base="type" >  
    ...  
  </restriction>  
</simpleType>
```

# Types simples – Restrictions (2/4)

- ◆ La définition de l'ensemble des restrictions est disponible à [www.w3.org/TR/xmlschema-0/#SimpleTypeFacets](http://www.w3.org/TR/xmlschema-0/#SimpleTypeFacets)
  - maxExclusive : limite supérieure (exclue)
  - maxInclusive : limite supérieure (incluse)
  - minExclusive : limite inférieure (exclue)
  - minInclusive : limite inférieure (incluse)
  - enumeration : liste de valeurs autorisée
  - length : nombre de caractères ou d'élément d'une liste autorisé
  - minLength : nombre minimum de caractères ou d'élément d'une liste
  - pattern : expression régulière à respecter
  - fractionDigits : nombre maxi de digits autorisé
  - totalDigits : nombre exact de digits autorisé
  - whiteSpace : politique de gestion des espaces blancs (tab, retour
  - ligne, fin de ligne, ...)

# Types simples – Restrictions (3/4)

## Exemple 1

Restriction sur des valeurs

```
<simpleType name="ageType">  
  <restriction base="integer">  
    <minInclusive value="1" />  
    <maxInclusive value="100" />  
  </restriction>  
</simpleType>
```

## Exemple 2

Restriction sur une liste valeur autorisée

```
<simpleType name="sexeTypeChild" >  
  <restriction base="string">  
    <enumeration value="homme" />  
    <enumeration value="femme" />  
    <enumeration value="indéterminé" />  
  </restriction>  
</simpleType>
```

## Exemple 3

Restriction à partir d'une expression régulière

```
<simpleType name="emailType">  
  <restriction base="string">  
    <pattern value=".*@.*" />  
  </restriction>  
</simpleType>
```

# Types simples – Restrictions (4/4)

- ◆ L'utilisation d'un nouveau type impose d'exploiter l'espace de nommage du Schema XML en cours
- ◆ La création d'un nouveau type peut être anonyme, de ce fait aucune valeur n'est précisée dans l'attribut name de la balise simpleType
- ◆ Une déclaration d'un type anonyme doit se faire dans le corps d'un élément ou d'un attribut simple
- ◆ L'utilisation de types anonymes ne permet pas de réutiliser les nouveaux types définis, ils ne sont utilisables qu'une seule fois
- ◆ Exemple:

```
<element name="old" />
  <simpleType>
    <restriction base="integer">
      <minInclusive value="1" />
      <maxInclusive value="100" />
    </restriction>
  </simpleType>
</element>
```

# Les attributs

- ◆ l'attribut est toujours déclaré comme un type simple
- ◆ Syntaxe d'un attribut de type simple

```
<attribut name="theName" type="theType" use="required" />
```

- ◆ L'élément attribut d'un Schema XML peut avoir trois attributs optionnels : use, default et fixed.
  - Un attribut est optionnel par défaut, pour préciser qu'il est obligatoire, il faut utiliser l'attribut use
  - Les attributs peuvent avoir une valeur par défaut (l'attribut default) ou une valeur fixée (l'attribut fixed)
- ◆ Par exemple, la ligne suivante permet de rendre l'attribut maj optionnel, avec une valeur par défaut au 11 octobre 2003

```
<xsd:attribute name="maj" type="xsd:date" use="optional" default="2003-10-11" />
```

# Les attributs

DTD	Attribut use	Attribut default
#REQUIRED	required	-
"val" #REQUIRED	required	val
#IMPLIED	optional	-
"val" #IMPLIED	optional	val



# Types complexes - généralités

- ◆ Un élément de type complexe peut contenir d'autres éléments et / ou des attributs
- ◆ Un attribut ne peut être de type complexe
- ◆ Quatre combinaisons d'éléments complexes sont à distinguer
  - Éléments vides qui ne contiennent que des attributs
  - Éléments de type simple qui contiennent des attributs
  - Éléments qui peuvent contenir des sous éléments
  - Éléments qui peuvent contenir des sous éléments et des attributs
- ◆ La création d'un éléments de type complexe est réalisée au travers de la balise `<complexType>`

# Types complexes - Uniquement attributs

- ◆ Il s'agit d'un élément de type complexe ne contenant que des attributs, appelé également élément vide
- ◆ Exemple

```
<xs:element name="child">  
  <xs:complexType>  
    <attribute name="remark" type="string" use="required" />  
    <attribute name="old" type="tns:ageTypeChild" />  
    <attribute name="sexe" type="tns:sexeTypeChild" />  
  </xs:complexType>  
</xs:element>
```

Dans une instance XML, la balise child contient 3 attributs et le corps de l'élément child est vide

```
<tns:child tns:remark="He's too much" tns:old="3" tns:sexe="homme"/>
```

# Types complexes – Une simple valeur et des attributs (1/2)

- ◆ Il s'agit d'un élément de type complexe qui peut contenir une simple valeur dans son corps et des attributs

- ◆ Exemple:

```
<tns:child tns:old="3" tns:sexe="homme" >Tom</tns:child>
```

- ◆ Le contenu de la balise simpleContent précise qu'il ne peut y avoir que des attributs et une simple valeur
- ◆ Deux écritures selon si vous souhaitez une restriction ou une extension

```
<xs:element name="">  
  <xs:complexType>  
    <xs:simpleContent>  
      <xs:extension base="baseType">  
        ...  
      </xs:extension>  
    </xs:simpleContent>  
  </xs:complexType>  
</xs:element>
```

```
<xs:element name="person">  
  <xs:complexType>  
    <xs:simpleContent>  
      <xs:restriction base="baseType">  
        ...  
      </xs:restriction>  
    </xs:simpleContent>  
  </xs:complexType>  
</xs:element>
```

# Types complexes – Une simple valeur et des attributs (2/2)

- ◆ Une extension (restriction a été vue précédemment) permet d'employer un type sans définition de contraintes
- ◆ Exemple

```
<tns:child tns:old="3" tns:sexe="homme" >Tom</tns:child>
```

```
<xs:element name="child">  
  <xs:complexType>  
    <xs:simpleContent>  
      <xs:extension base="xs:string">  
        <attribute name="old" type="tns:ageTypeChild" />  
        <attribute name="sexe" type="tns:sexeTypeChild" />  
      </xs:extension>  
    </xs:simpleContent>  
  </xs:complexType>  
</xs:element>
```

# Les types complexes - Sous éléments

- ◆ Définition d'objets complexes
  - `<xs:sequence>` : collection ordonnée d'éléments typés
  - `<xs:all>` : collection non ordonnée d'éléments typés
  - `<xs:choice>` : choix entre éléments typés
- ◆ Exemple: Soit le document xml suivant, écrire le fichier xsd correspondant

```
<Adresse>  
<nom> ..... </nom>  
<rue>.....</rue>  
<codep>.....</codep>  
</Adresse>
```

# Types complexes – Sous éléments (Séquences)

- ◆ Dans une DTD, nous pouvons déclarer un élément comme pouvant contenir une suite de sous-éléments dans un ordre déterminé
- ◆ On utilise l'élément `xsd:sequence`, qui reproduit l'opérateur **, du langage DTD**

```
<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>
```

# Types complexes – Sous éléments (Choix)

- ◆ Si on indique soit l'adresse d'une personne, soit son adresse électronique. il suffit d'utiliser un élément `xsd:choice`
- ◆ Ce connecteur a donc les mêmes effets que l'opérateur **|** **dan** une DTD.

```
<xsd:complexType name="typePersonne">
  <xsd: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:sequence>
</xsd:complexType>
```

# Types complexes – Sous éléments (all)

- ◆ La balise all spécifie que tous les sous éléments peuvent apparaître dans n'importe quel ordre

```
<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>
```



# Types complexes – Sous éléments (occurrences)

- ◆ Les indicateurs d'occurrence sont utilisés pour exprimer le
- ◆ nombre de fois qu'un sous élément peut apparaître
- ◆ Ils sont exprimés sous la forme d'attributs d'un sous élément
  - maxOccurs : précise le nombre d'occurrence maximum
  - minOccurs : précise le nombre d'occurrence minimum
- ◆ Si les valeurs de maxOccurs ou minOccurs ne sont pas explicitement précisées, la valeur par défaut est de 1
- ◆ Pour définir une valeur infinie, fixer la valeur à unbounded
- ◆ Exemple : Une bibliothèque contient au moins un livre

```
<xs:element name="biblio">  
  <xs:complexType>  
    <xs:sequence>  
      <xs:element name="livre" minOccurs="1" maxOccurs="unbounded"/>  
    </xs:sequence>  
  </xs:complexType>  
</xs:element>
```

# Types complexes – Sous éléments (occurrences)

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

# Types complexes – sous éléments et des attributs

- ◆ Un élément de type complexe contenant des sous éléments peut également contenir des attributs

```
<element name="person">  
  <complexType>  
    <sequence>  
      <element name="name" type="string" />  
      <element name="firstName" type="string" />  
      <element name="old" type="tns:ageType" />  
      <element name="email" type="tns:ageType" />  
    </sequence>  
    <attribute name="userId" type="string" />  
  </complexType>  
</element>
```

# Types complexes : complexContent

- ◆ Possibilité de définir un nouveau type complexe sur la base d'un type complexe existant de manière à étendre ou à restreindre les éléments
- ◆ Utilisation de la balise `complexContent`
- ◆ Exemple

```
<complexType name="Address">  
  <sequence>  
    <element name="name" type="string" />  
    <element name="street" type="string" />  
    <element name="city" type="string" />  
  </sequence>  
</complexType>
```

Le type `Address` définit  
trois sous éléments en  
séquence

```
<complexType name="USAddress">  
  <complexContent>  
    <extension base="tns:Address">  
      <sequence>  
        <element name="state" type="string" />  
        <element name="zip" type="string" />  
      </sequence>  
    </extension>  
  </complexContent>  
</complexType>
```

Le type `USAddress` propose une extension  
avec deux nouveaux sous éléments en séquence

# Exercice

- ◆ On se propose de définir un format XML qui est utilisé pour décrire une personne
- ◆ Soit les contraintes suivantes qui devront être respectées
  - Une personne doit définir un nom puis un prénom puis un âge et peut définir un email puis des renseignements sur ces enfants
  - L'âge est un entier compris entre 1 et 100
  - L'email est une chaîne de caractères qui doit contenir le caractère @
  - Les renseignements concernant les enfants portent sur l'âge (obligatoire), le sexe (obligatoire) et une remarque (facultative)
  - L'âge de l'enfant est une restriction de l'âge de la personne dont la valeur maxi est modifiée à 50
  - Le sexe est un énuméré (homme, femme et indéterminé)
  - Il ne peut y avoir plus de 3 enfants

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3         targetNamespace="http://mbaron.developpez.com/InfoPersoSchema"
4         xmlns:tns="http://mbaron.developpez.com/InfoPersoSchema"
5         elementFormDefault="qualified"
6         attributeFormDefault="qualified">
7     <element name="person">
8         <complexType>
9             <sequence>
10                <element name="name" type="string" />
11                <element name="firstName" type="string" />
12                <element name="old" type="tns:ageType" />
13                <element name="email" type="tns:emailType"
14                    minOccurs="0" />
15                <element name="childs" type="tns:childsType"
16                    minOccurs="0" maxOccurs="1" />
17            </sequence>
18        </complexType>
19    </element>
20
21    <simpleType name="ageType">
22        <restriction base="integer">
23            <minInclusive value="1" />
24            <maxInclusive value="100" />
25        </restriction>
26    </simpleType>
27
28    <simpleType name="emailType">
29        <restriction base="string">
30            <pattern value=".*@.*" />
31        </restriction>
32    </simpleType>
```

```
33
34 <simpleType name="sexeTypeChild" >
35     <restriction base="string">
36         <enumeration value="homme" />
37         <enumeration value="femme" />
38         <enumeration value="indÃ©terminÃ©" />
39     </restriction>
40 </simpleType>
41
42 <simpleType name="ageTypeChild">
43     <restriction base="tns:ageType">
44         <maxInclusive value="50" />
45     </restriction>
46 </simpleType>
47
48 <complexType name="childsType">
49     <sequence>
50         <element name="child" minOccurs="1" maxOccurs="3">
51             <complexType>
52                 <attribute name="remark" type="string"
53                     use="required" />
54                 <attribute name="old" type="tns:ageTypeChild" />
55                 <attribute name="sexe" type="tns:sexeTypeChild" />
56             </complexType>
57         </element>
58     </sequence>
59 </complexType>
60 </schema>
```



# **ANNEXE**



# XML-Namespace

- ◆ [Définition :] Un **espace de nommage XML** est une collection de noms, identifiée par une référence d'URI qui sont utilisés dans les documents XML comme types d'élément et noms d'attribut

# The schema element defines the root element of a schema

Attribute	Description
<b>id</b>	Optional. Specifies a unique ID for the element
<b>attributeFormDefault</b>	Optional. The form for attributes declared in the target namespace of this schema. The value must be "qualified" or "unqualified". Default is "unqualified". "unqualified" indicates that attributes from the target namespace are not required to be qualified with the namespace prefix. "qualified" indicates that attributes from the target namespace must be qualified with the namespace prefix
<b>elementFormDefault</b>	Optional. The form for elements declared in the target namespace of this schema. The value must be "qualified" or "unqualified". Default is "unqualified". "unqualified" indicates that elements from the target namespace are not required to be qualified with the namespace prefix. "qualified" indicates that elements from the target namespace must be qualified with the namespace prefix
<b>blockDefault</b>	<ul style="list-style-type: none"><li>•Optional. Specifies the default value of the block attribute on element and complexType elements in the target namespace. The block attribute prevents a complex type (or element) that has a specified type of derivation from being used in place of this complex type. This value can contain #all or a list that is a subset of extension, restriction, or substitution: extension - prevents complex types derived by extension</li><li>•restriction - prevents complex types derived by restriction</li><li>•substitution - prevents substitution of elements</li><li>•#all - prevents all derived complex types</li></ul>
<b>finalDefault</b>	<ul style="list-style-type: none"><li>•Optional. Specifies the default value of the final attribute on element, simpleType, and complexType elements in the target namespace. The final attribute prevents a specified type of derivation of an element, simpleType, or complexType element. For element and complexType elements, this value can contain #all or a list that is a subset of extension or restriction. For simpleType elements, this value can additionally contain list and union: extension - prevents derivation by extension</li><li>•restriction - prevents derivation by restriction</li><li>•list - prevents derivation by list</li><li>•union - prevents derivation by union</li><li>•#all - prevents all derivation</li></ul>
<b>targetNamespace</b>	Optional. A URI reference of the namespace of this schema
<b>version</b>	Optional. Specifies the version of the schema
<b>xmlns</b>	A URI reference that specifies one or more namespaces for use in this schema. If no prefix is assigned, the schema components of the namespace can be used with unqualified references
<b>any attributes</b>	Optional. Specifies any other attributes with non-schema namespace