

Technologies de Schemas XML



Qu'est-ce qu'un schéma ?

Document Type Definition

- Modèles de contenus
- Déclarations d'attributs
- Entités générales / paramètres, internes / externes
- Appels de DTD
- Identificateurs et catalogues
- Documents bien formés, réglages de l'analyse
- Conception des DTD : best practice

W3C XML Schema

(Présentation)

- Types de données
- Structures
- Modèles de contenus
- Attributs
- Autres structures

Relax NG

(Présentation)

- Structures
- Modèles de contenus
- Attributs
- Autres structures
- Typage avec WXS

PSVI et validation

PSVI

Choisir une techno, incomplétude de la validation
Schematron

Les DTD appartiennent à une classe fonctionnelle de documents : les schémas

Les schémas décrivent la grammaire utilisable dans un document XML

Document Type Definition	DTD	syntaxe non XML	W3C
W3C XML Schema	WXS	syntaxe XML	
	Relax-NG		ISO

D'autres propositions ont été faites, les offres ne manquent pas...

Microsoft XDR (*XML-Data Reduced*) est une implémentation basée sur les travaux d'origine du W3C (*XML-Data Note* et *Document Content Description (DCD) initiative for XML*)

Document Type Definition

Document Type Definition

DTD : description formelle de la structure du document

Une DTD peut être :

- locale au document (sous-ensemble interne),
- importée dans le document (sous-ensemble externe)
- ou les deux

Permet de contraindre
les données d'un
document

Fait partie du standard
XML, ne surcharge pas
les applications

Une DTD contient (entre autres) :

- La définition des éléments-type
- Leurs listes d'attributs
- La définition des entités → assimilable à des macros-textes

La DTD est facultative

Mais :

- Un document qui déclare une DTD doit s'y conformer
 - Dans un contexte professionnel, on ne peut pas s'en passer
- Pour « bricoler », on peut s'en passer



Validation

Intérêt d'une DTD externe : permet d'être réutilisée par d'autres documents

Utilisation optimale :

- sous-ensemble externe : pour la classe de document
- sous-ensemble interne : pour des particularités propres au document

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Hotels -->
<!ELEMENT City (#PCDATA) <!-- Also used in Address -->
<!ELEMENT Island (#PCDATA)>
<!ELEMENT Location (City, Island)>
<!ELEMENT Hotel (Location)>
<!ATTLIST Hotel
    Id ID #REQUIRED
    Name CDATA #REQUIRED
    Stars (1 | 2 | 3 | 4 | 5) #REQUIRED
>
<!ELEMENT Hotels (Hotel+)>

<!-- Offers -->
<!ELEMENT Flight EMPTY>
<!ATTLIST Flight
    Id ID #REQUIRED
    Price CDATA #REQUIRED
    DepartureAirport NMTOKEN #REQUIRED
    DepartureTime CDATA #REQUIRED
    ArrivalAirport NMTOKEN #REQUIRED
    ArrivalTime CDATA #REQUIRED
    DayOffset CDATA "0"
>
<!ELEMENT Offers ((Flight+, Stay*, Tour*) | (Stay+, Tour*) | (Tour+))>

.../...
```

Un élément peut se contenir

Syntaxe `<!ELEMENT nom-element (modèle-de-contenu)>`

Modèles de contenu :

Séquence :

- indique quel(s) élément(s) compose(nt) le modèle
- liste d'éléments séparés par une virgule
- tous les éléments devront apparaître dans l'ordre prévu

Pour spécifier qu'un Auteur doit avoir un Nom puis un Prénom :

Exemple

`<!ELEMENT Auteur (Nom, Prénom)>`



```
<Auteur>
  <Nom>Poulard</Nom>
  <Prénom>Philippe</Prénom>
</Auteur>
```

Element ne contenant que des données textuelles parsées

Syntaxe `<!ELEMENT nom-element (#PCDATA)>`

(Parsed Character Data)

Exemple

`<!ELEMENT Nom (#PCDATA)>`

Element vide

Syntaxe `<!ELEMENT nom-element EMPTY>`

Exemples

`<!ELEMENT br EMPTY>`

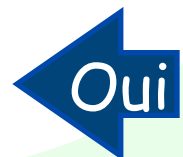
`<!ELEMENT img EMPTY>`



```
<br>      </br>
<br><!--></br>
<br><?br?></br>
```



```
<br></br>
<br/>
```



Modèles de contenu :

Choix :

- indique quel(s) élément(s) compose(nt) le modèle
- liste d'éléments séparés par une barre verticale
- **un seul élément de la liste devra apparaître**

Pour spécifier qu'un Menu peut contenir un Fromage ou un Dessert, mais pas les deux :

Exemple

```
<!ELEMENT Menu (Fromage | Dessert)>
```



```
<Menu>
  <Fromage>Camembert</Fromage>
</Menu>
```

Modèle de contenu universel :

Permet à un sous-élément de contenir n'importe quoi (éléments et contenu)
Les éléments qu'il peut contenir doivent être déclarés

Syntaxe

```
<!ELEMENT nom-element ANY>
```

Très peu utilisé, très peu pratique

Modèles de contenu combinés :

Choix et séquences peuvent être combinés à volonté
Les regroupements se font avec des parenthèses

Pour spécifier qu'un Menu est un peu plus compliqué...

Exemple

```
<!ELEMENT Menu (Entrée, Plat, (Fromage | Dessert))>
```


Fréquence d'apparition d'un sous-élément

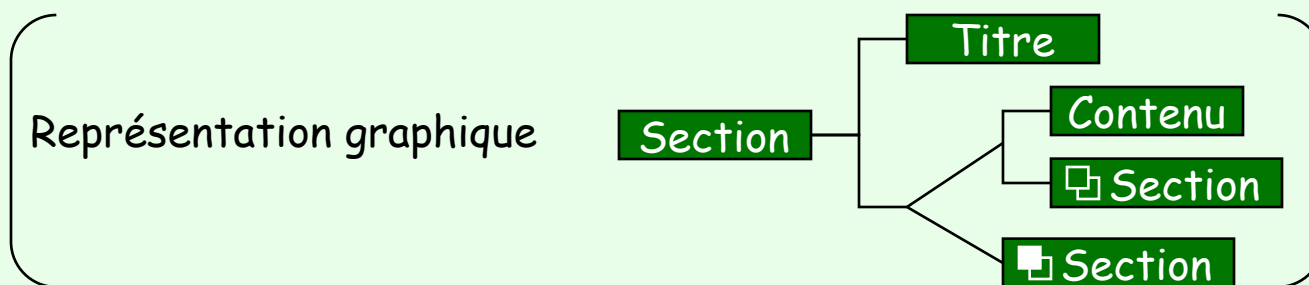
Un suffixe indique le nombre d'instance attendues
 Peut s'appliquer à un élément seul ou à un groupe d'éléments

Suffixe	Cardinalité	Description
	(1, 1)	Obligatoire (par défaut) : doit apparaître une et une seule fois
?	(0, 1)	Optionnel : doit apparaître zéro ou une seule fois
*	(0, n)	Multiple : peut apparaître plusieurs fois ou ne pas apparaître
+	(1, n)	Multiple obligatoire : doit apparaître au moins une fois

Exemples

```
<!ELEMENT Menu (Entrée?, Plat, (Fromage* | Dessert))>
```

```
<!ELEMENT Section (Titre, ((Contenu, Section*) | Section+))>
```



```
<!ELEMENT foo (#PCDATA , bar?)>
```

Non

```
<!ELEMENT foo (#PCDATA | bar)*>
```

Oui



```
<foo>Il y a à la fois du contenu <bar>et un élément</bar>.</foo>
```

Dans tout model contenant #PCDATA et d'autres éléments :

- ce doit être un groupement simple séparé par "|"
- #PCDATA doit apparaître en premier
- le groupe doit être optionnel et répétable (*)

Solution alternative :

```
<!ELEMENT text (#PCDATA)>
<!ELEMENT foo (text, bar?)>
```



Eviter de définir des structures avec du contenu mixte

→ difficilement réalisable dans les structures documentaires

Syntaxe

```
<!ATTLIST nom-element
      nom-attribut type défaut
      nom-attribut type défaut
      nom-attribut type défaut
>
```

Déclaration pour
chaque attribut

{ ...qui peut être faite
séparément }

Une déclaration d'attribut est réalisée au profit d'un élément

Tous les attributs d'un élément doivent être déclarés

(y compris les attributs spéciaux `xmlns`, `xml:space` et `xml:lang`)

On ne peut pas spécifier l'ordre d'apparition des attributs dans un élément

Exemple

	Type	Défaut
<code><!ATTLIST image</code>	<code>source</code>	<code>CDATA #REQUIRED</code>
	<code>hauteur</code>	<code>CDATA #REQUIRED</code>
	<code>largeur</code>	<code>CDATA #REQUIRED</code>
<code>></code>	<code>texte</code>	<code>CDATA #IMPLIED</code>

Type *character data* (CDATA)

Présence de l'attribut obligatoire

```
<image source="bubulle.jpg"
       largeur="10cm"
       hauteur="5cm"
       texte="mon poisson rouge"/>
```

Types d'attributs possibles :

- CDATA**
- NMTOKEN
- ID
- NOTATION
- NMTOKENS
- IDREF
- Énumération
- ENTITY
- IDREFS
- ENTITIES

Les valeurs des attributs peuvent
contenir des appels d'entités

Les retours chariots contenus dans
les valeurs d'attributs ne sont pas
significatifs

CDATA est le type le plus général, la
valeur de l'attribut peut contenir
n'importe quelle chaîne de caractères

- CDATA
- NOTATION
- Enumération
- NMTOKEN
- NMTOKENS
- ENTITY
- ENTITIES
- ID
- IDREF
- IDREFS

La valeur d'un attribut de type `NMTOKEN` doit contenir des caractères d'unité **lexicale nominale**

Caractères alphabétiques accentués
Chiffres
Caractères `_ - . :`
Caractères des autres langues non romanes

Exemple

```
<!ATTLIST user      insee      NMTOKEN  #REQUIRED>
```



```
<user insee="1-69-06-13-001-084"/>
```

La valeur d'un attribut de type `NMTOKENS` contient une ou plusieurs **unités lexicales XML** séparées par des blancs

Exemple

```
<!ATTLIST article  nno      NMTOKENS #REQUIRED>
```



```
<article nno="1234 14 1234567"/>
```

- CDATA
- NOTATION
- **Enumération**
- NMTOKEN
- NMTOKENS
- ENTITY
- ENTITIES
- ID
- IDREF
- IDREFS

Les valeurs possibles que peut prendre un attribut peuvent être énumérées

Syntaxe

```
<!ATTLIST nom-element nom-attribut (val1 | val2 | val3 ... | valn) default>
```

Exemples

```
<form method="post"/>
```

Défaut



```
<!ATTLIST form      method      ( get | post )      'get'>
```

```
<!ATTLIST date      mois          ( janvier | février | mars | avril | mai
| juin | juillet | août | septembre
| octobre | novembre | decembre      )      #REQUIRED
>
```

```
<date mois="juin"/>
```

Chaque valeur de la liste doit être une unité lexicale nominale
 (→ aucune valeur ne peut contenir de caractères blancs)

- CDATA
- NOTATION
- Enumération
- NMTOKEN
- NMTOKENS
- ENTITY
- ENTITIES
- ID
- IDREF
- IDREFS

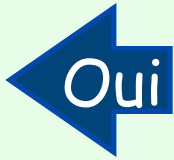
La valeur d'un attribut de type ID doit être un **nom XML**, unique dans le document (2 attributs de type ID ne peuvent pas avoir la même valeur)

- Un élément ne peut définir qu'un seul attribut de type ID
- Un attribut dont le nom est id n'est pas nécessairement de type ID
- Le nommage d'un attribut de type ID n'est pas plus contraint que les autres attributs

```
<!ATTLIST a id ID #IMPLIED
           name ID #IMPLIED >
```



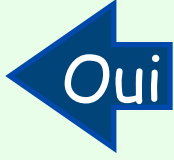
```
<!ATTLIST a id ID #IMPLIED
           name NMTOKEN #IMPLIED >
```



```
<a href="fichier1.htm" id="link">fichier 1</a>
<a href="fichier2.htm" id="link">fichier 2</a>
```



```
<a href="fichier1.htm" id="link1">fichier 1</a>
<a href="fichier2.htm" id="link2">fichier 2</a>
```



- Un attribut de type IDREF contient la valeur d'un attribut de type ID
- Un attribut de type IDREFS en contient une ou plusieurs séparées par des blancs

```
<!ELEMENT user (#PCDATA)>
<!ATTLIST user id ID #REQUIRED>

<!ELEMENT project (#PCDATA)>
<!ATTLIST project users IDREFS #REQUIRED>
```



```
<user id="u1"/>Jean</user>
<user id="u2"/>Paul</user>
<user id="u3"/>Marc</user>
<project users="u1 u2">Foo</project>
<project users="u2 u3">Bar</project>
<project users="u1 u2 u3">Any</project>
```

- CDATA
- **NOTATION**
- Enumération
- NMTOKEN
- NMTOKENS
- ENTITY
- ENTITIES
- ID
- IDREF
- IDREFS

La valeur d'un attribut de type NOTATION est le nom d'une des notations déclarées dans la DTD et énumérées pour l'attribut

Exemple

```
<!NOTATION jpeg SYSTEM "image/jpeg">
<!NOTATION gif SYSTEM "image/gif">
<!NOTATION png SYSTEM "image/png">
<!ATTLIST image source CDATA #REQUIRED
hauteur CDATA #REQUIRED
largeur CDATA #REQUIRED
texte CDATA #IMPLIED
type NOTATION (gif | jpeg | png) #REQUIRED
>
```



```
<image source="photo.jpg"
largeur="10cm"
hauteur="5cm"
texte="mon poisson rouge"
type="jpeg" />
```

...d'un usage très limité (héritage de SGML)

- CDATA
- NOTATION
- Enumération
- NMTOKEN
- NMTOKENS
- ENTITY
- ENTITIES
- ID
- IDREF
- IDREFS

La valeur d'un attribut de type ENTITY est le nom d'une entité non parsée déclarée dans la DTD

Exemple

```
<!NOTATION jpeg SYSTEM "image/jpeg">
<!ENTITY photo SYSTEM "photo.jpg" NDATA jpeg>
<!ATTLIST image source ENTITY #REQUIRED
                 hauteur CDATA #REQUIRED
                 largeur CDATA #REQUIRED
                 texte CDATA #IMPLIED
>
```

C'est un moyen intégré au standard qui permet d'indiquer à une application que d'autres fichiers (images...) font partie du document.

Utilisé par des éditeurs (EPIC)

```
<image source="photo"
        largeur="10cm"
        hauteur="5cm"
        texte="mon poisson rouge"/>
```

• Un attribut de type ENTITIES contient un ou plusieurs noms d'entités séparés par des blancs

Exemple

```
<!ATTLIST diaporama diapos ENTITIES #REQUIRED>
```

```
<diaporama diapos="intro objectifs résultats analyse outro"/>
```


#IMPLIED

Attribut optionnel

#REQUIRED

Attribut obligatoire

```
<!ATTLIST image      source      CDATA      #REQUIRED
                   hauteur     CDATA      #REQUIRED
                   largeur     CDATA      #REQUIRED
                   texte       CDATA      #IMPLIED
>
```

Littéral

Valeur par défaut en tant que chaîne entre quotes (simples / doubles)
L'application doit considérer l'attribut renseigné avec cette valeur
s'il n'est pas noté dans le document

```
<!ATTLIST form      method      ( get | post )      'get'>
```

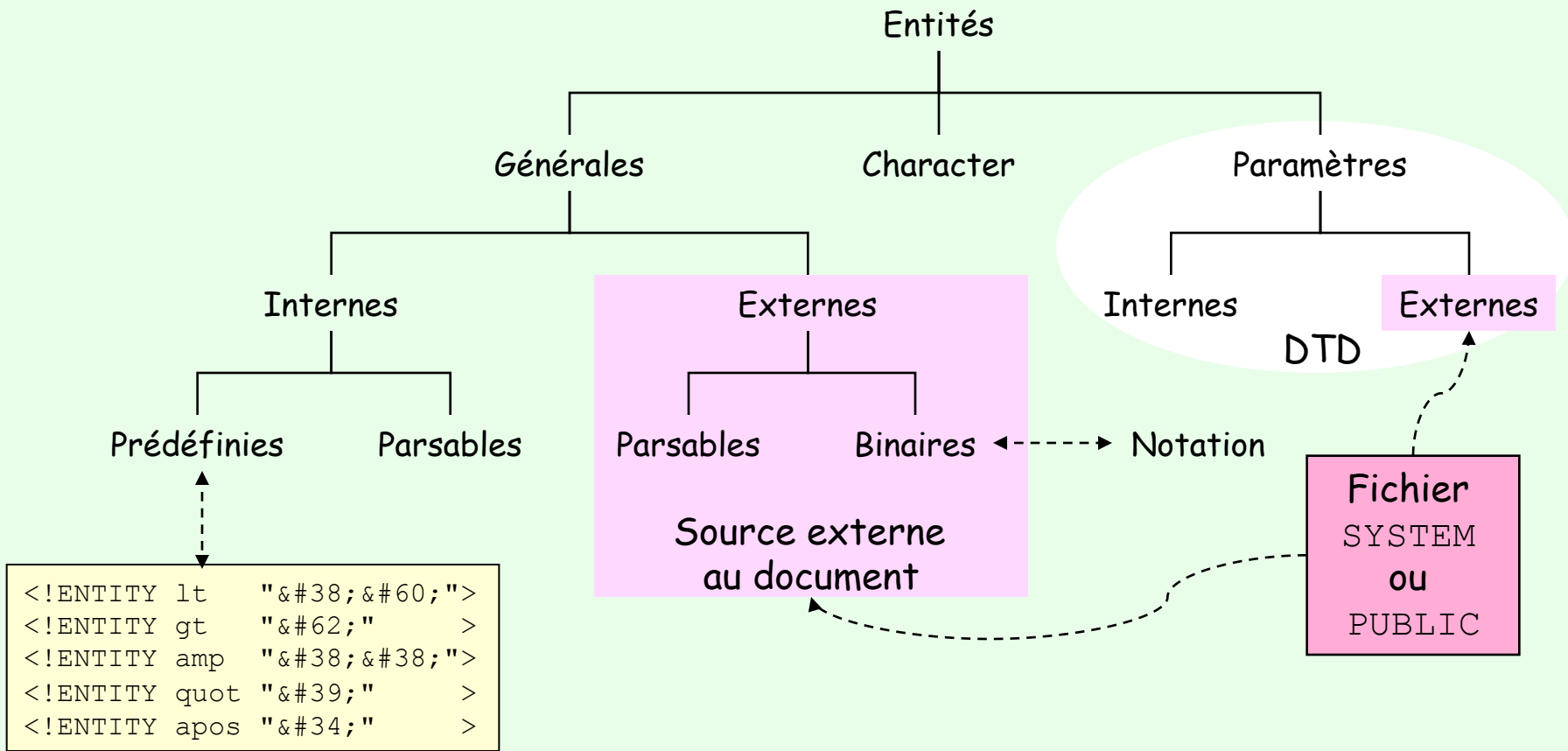
#FIXED

Valeur fixe et non modifiable

Bien que l'attribut puisse ne pas être explicitement noté dans le document, l'application doit considérer l'attribut comme renseigné

```
<!ATTLIST svg      xmlns      CDATA      #FIXED      "http://www.w3.org/2000/svg">
```

```
<!ATTLIST livre     xmlns:xlink CDATA      #FIXED      "http://www.w3.org/1999/xlink">
```



Le contenu de remplacement d'une entité parsée est du texte
 Une entité interne DOIT être une entité parsable

L'instance XML est aussi appelée l'**entité document**

Qui peut contenir des sections CDATA (<! [CDATA []]>), donc non parsée

Déclaration

• dans la DTD

```
<!ENTITY version "2.1">
```

Caractères interdits :

% & "

(ils doivent être échappés par des appels de caractères)

Appel

```
<pub>
Commandez la dernière version (&version;)
</pub>
```

Rendu

Commandez la dernière version (2.1)

Déclaration avec éléments (biens formés)

```
<!ENTITY flipper
"<dauphin photo='flipper.jpg'>
  <nom>Flipper</nom>
  <age>12</age>
</dauphin>">
```

Appel dans la DTD

```
<!ENTITY release "1">
<!ENTITY version "2.&release;">
```



Restrictions:

• pas de référence circulaire

```
<!ENTITY a "&b;">
<!ENTITY b "&a;">
```

Non

• pas de possibilité d'insérer du contenu qui fait partie de la DTD (les entités paramètres servent à ça)

```
<!ENTITY a "#PCDATA">
<!ELEMENT b &a;>
```

Non

Externes = dans un autre fichier que le document

Les entités externes sont référencées par un **identificateur** :

- SYSTEM : URI vers le nom du fichier
- PUBLIC : référence dans un catalogue connu du parseur
- ou les 2

Les **identificateurs** ne sont pas parsés

```
<!ENTITY acme SYSTEM "http://www.acme.com/entities/MyEntity.xml">
```

```
<!ENTITY acme PUBLIC "-//ACME//My entity//EN">
```

```
<!ENTITY acme PUBLIC "-//ACME//My entity//EN"  
"http://www.acme.com/entities/MyEntity.xml">
```

Les résolutions d'entités sont effectuées au moment de leur appel

L'appel d'une entité externe parsée ne peut se faire que dans le contenu d'un élément



```
<foo bar="&acme ;"/>
```

Non

```
<foo>&acme ;</foo>
```

Oui

MyEntity.xml

```
<truc>  
Du contenu avec <machin chose="des  
balise">et des attributs</machin>  
</truc>
```



Les parseurs validants **DOIVENT** remplacer les appels d'entité par leur contenu
Les parseurs non validants peuvent ne pas rapatrier les entités externes parsées

L'entité externe peut ne pas être un document XML bien formé (en particulier, il n'est pas nécessaire qu'il ait un seul élément racine), du moment que le document qui l'appelle soit lui, un document bien formé.

Tout élément ouvert dans l'entité externe doit se fermer dans cette même entité.

Declaration de texte

Une entité externe n'a ni prologue ni déclaration de DTD

Toutefois, une entité externe peut contenir une déclaration de texte

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

ou

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

Chaque entité peut avoir son propre encodage différent de l'entité document maître.

Une entité externe n'a pas de DTD, mais peut être l'unique contenu d'un document qui peut avoir sa DTD (qui correspond à l'entité seulement)

```
<?xml encoding="ISO-8859-1"?>
<!DOCTYPE foo SYSTEM foo.dtd [
<!ENTITY bar SYSTEM bar.xml>
]>
<foo>Document complet avec
des appels à plusieurs structures
importées
&bar;
</foo>
```

1-Ce fichier incorpore une entité externe

```
<?xml encoding="ISO-8859-1"?>
<!DOCTYPE bar SYSTEM bar.dtd[
<!ENTITY bar SYSTEM bar.xml>
]>
&bar;
```

3-L'entité externe peut être contrainte par un fichier maître qui l'appelle : l'entité bar.xml peut ainsi avoir sa propre DTD

bar.xml

```
<?xml encoding="ISO-8859-1"?>
<bar>Contenu contraint par une DTD
</bar>
```

2-Ce fichier n'a pas de DTD car il est incorporé dans un document qui est déjà doté de sa DTD

Une meilleure méthode :
XInclude

<http://www.w3.org/TR/xinclude/>

```
<?xml encoding="ISO-8859-1"?>
<!DOCTYPE foo SYSTEM foo.dtd>
<foo>Document complet avec
des appels à plusieurs structures
importées
<xi:include href="bar.xml"
xmlns:xi="http://www.w3.org/2001/XInclude"/>
</foo>
```

Déclaration dans la DTD

```
<!NOTATION jpeg SYSTEM "image/jpeg">
<!ENTITY photo SYSTEM "photo.jpg" NDATA jpeg>
<!ATTLIST image source ENTITY #REQUIRED
                 hauteur CDATA #REQUIRED
                 largeur CDATA #REQUIRED
                 texte CDATA #IMPLIED
>
```

C'est un moyen intégré au standard qui permet d'indiquer à une application que d'autres fichiers (images...) font partie du document.

↓
Utilisé par des éditeurs (EPIC)

Appel dans le document

```
<image source="photo"
        largeur="10cm"
        hauteur="5cm"
        texte="mon poisson rouge"/>
```

```

<!NOTATION jpeg      SYSTEM "image/jpeg">
<!ATTLIST  image     source      ENTITY  #REQUIRED
           hauteur    CDATA      #REQUIRED
           largeur    CDATA      #REQUIRED
           texte      CDATA      #IMPLIED
>
<!ENTITY   photo     SYSTEM "photo.jpg" NDATA jpeg>

```

devrait être défini dans le sous-ensemble interne de la DTD (l'image fait partie du document)

Dépendance applicative

```
<!NOTATION jpeg SYSTEM "apps/imgviewer.exe">
```

À éviter : dépendance vis à vis d'une application donnée sur un système donné

Formel

```

<!NOTATION jpeg PUBLIC "ISO/IEC 10918:1993//NOTATION
    Digital Compression and Coding of
    Continuous-tone Still Images
    (JPEG)//EN" >

```



MIME

```
<!NOTATION jpeg SYSTEM "image/jpeg">
```

À la mode...

Déclaration : `<!ENTITY % foo "bar">`

Appel : `%foo;`

Différence avec les entités générales :

- Appel avec le signe % au lieu de &
- Appel uniquement dans la DTD et jamais dans le document
- Permet de **factoriser** les définitions d'élément

Contraintes :

- Une entité paramètre doit toujours être déclarée **avant** d'être appelée
- Une entité paramètre peut être appelée pour **compléter** une déclaration uniquement dans le sous-ensemble externe

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE zoo SYSTEM "zoo.dtd">
```

zoo.dtd

```
<!ENTITY % animal "nom, sexe, taille, poids, date-naissance, commentaire?">
<!ELEMENT dauphin (%animal;)>
```

Oui

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE zoo [
<!ENTITY % animal "nom, sexe, taille, poids, date-naissance, commentaire?">
<!ELEMENT dauphin (%animal;)>
]>
```

Non

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE zoo [
<!ENTITY % HR "<!ELEMENT hr EMPTY">">
%HR;
]>
```



Déclaration et appel corrects
dans le sous-ensemble interne et
dans le sous-ensemble externe

```

<!ELEMENT dauphin (nom, sexe, taille, poids, date-naissance, commentaire?)>
<!ATTLIST dauphin
    id                ID #REQUIRED
    espèce            CDATA #IMPLIED
    nom-savant        CDATA #IMPLIED
    photo             ENTITY #IMPLIED
>
<!ELEMENT baleine (nom, sexe, taille, poids, date-naissance, commentaire?)>
<!ATTLIST baleine
    id                ID #REQUIRED
    espèce            CDATA #IMPLIED
    nom-savant        CDATA #IMPLIED
    photo             ENTITY #IMPLIED
>

```



```

<!ENTITY % animal "nom, sexe, taille, poids, date-naissance, commentaire?">
<!ENTITY % animal-attr "
    id                ID #REQUIRED
    espèce            CDATA #IMPLIED
    nom-savant        CDATA #IMPLIED
    photo             ENTITY #IMPLIED
">
<!ELEMENT dauphin (%animal;)>
<!ATTLIST dauphin %animal-attr;>
<!ELEMENT baleine (%animal;)>
<!ATTLIST baleine %animal-attr;>

```

Après substitution d'une entité paramètre par sa valeur, les parseurs peuvent insérer un espace avant et après, sauf si l'entité est contenue dans une chaîne

```

<![IGNORE [
  <!ELEMENT annotation (#PCDATA)>
  <!ELEMENT p (b | i | annotation)*>
]]>
<![INCLUDE [
  <!ELEMENT p (b | i)*>
]]>

```

Ces déclarations ne sont pas interprétées

← Cette déclaration est interprétée

Une section conditionnelle ne peut pas être utilisée pour **compléter** une déclaration

```

<!ELEMENT p (b | i <![INCLUDE[ | annotation]]>)*>

```

Non

Imbrications de blocs conditionnels :

```

<![IGNORE [
  <![IGNORE [
    .../...
  <![INCLUDE [
    .../...
  ]]]>
]]>
<![INCLUDE [
  <![IGNORE [
    .../...
  ]]]>
  <![INCLUDE [
    .../...
  ]]]>
]]>

```

Tout le bloc est ignoré, même le sous-bloc INCLUDE

Ce bloc est ignoré, bien qu'il soit dans un bloc INCLUDE

```
<![%extended-features;[
  <!-- pleins de déclarations en plus -->
]]>
```

```
<!ENTITY % extended-features "IGNORE">
```

```
<!ENTITY % extended-features "INCLUDE">
```

```
<![IGNORE[
  <!-- pleins de déclarations en plus -->
]]>
```

```
<![INCLUDE[
  <!-- pleins de déclarations en plus -->
]]>
```

```
<!ENTITY % full-dtd "IGNORE">
<!ENTITY % light-dtd "INCLUDE">
<![%full-dtd;[
  <!ELEMENT annotation (#PCDATA)>
  <!ENTITY % inline "b | i | annotation">
]]>
<![light-dtd;[
  <!ENTITY % inline "b | i">
]]>
<!ELEMENT p (%inline;)*>
```

```

<!ENTITY % ns-support SYSTEM "ns-support.ent">
<!ENTITY % xml-att SYSTEM "xml-att.ent">
<!ENTITY % svg PUBLIC "-//W3C//DTD SVG 1.0//EN" "http://www.w3.org/TR/2001/REC-SVG-20010904"
<!ENTITY % persons-mod SYSTEM "persons.mod">
<!-- inclusions -->
%ns-support;
%xml-att;
<![%SVG-IMPORT; [
    %svg;
]]>
<!-- elements definition -->
<!ENTITY % abstract &apos;%prefix;abstract&apos;>
<!ENTITY % answer &apos;%prefix;answer&apos;>
<!ENTITY % authors &apos;%prefix;authors&apos;>
<!ENTITY % b &apos;%prefix;b&apos;>

<!-- predefined sets -->
<!ENTITY % inline &apos;%stabilo; | %include; | %inverse; | %pc; | %i; | %b; | %a; | %big;
<!ENTITY % admonition-block &apos;%warning; | %note;&apos;>
<!ENTITY % doc-block &apos;%table; | %ul; | %ol; | %p; | %pre; | %sample; | %remarks;&apos;

<!-- root element -->
<!ENTITY % doc-root &apos;%prefix;document&apos;>
<!ELEMENT %doc-root; (%ident;, (%sources;)?, %content;)>
<!ATTLIST %doc-root;
    xmlns:xlink CDATA #FIXED "http://www.w3.org/1999/xlink"
    xmlns:xi CDATA #FIXED "http://www.w3.org/2001/XInclude"
>
<!-- persons -->
%persons-mod;
<!ELEMENT %persons; (%person;)+>
<!ELEMENT %authors; (%person;)+>
<!-- content structure -->
<!ELEMENT %content; (%flow;)>

```

```
<!DOCTYPE acme SYSTEM "http://www.acme.com/entities/MyDTD.dtd">
<!DOCTYPE acme PUBLIC "-//ACME//My DTD//EN"
"http://www.acme.com/entities/MyDTD.dtd">
```

Une seule déclaration par document
 Pour utiliser plusieurs DTD, faire des appels d'entités

Nom de l'élément racine du document

Les outils tentent de résoudre les identificateurs PUBLIC

- En accédant à un catalogue
- En utilisant un programme spécifique (EntityResolver)

S'ils n'y arrivent pas, ils accèdent au fichier SYSTEM

Les DTD peuvent être :

- externes
- internes
- ou les 2 : dans ce cas, le sous-ensemble interne est lu en premier

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE foo SYSTEM "foo.dtd" [
  <!ENTITY % FOO "INCLUDE">
  <!ENTITY bar.jpg SYSTEM "bar.jpg" NDATA jpg>
]>
<foo>
  .../...
</foo>
```

Peut utiliser des paramètres du sous-ensemble interne

Déclarations lues en premier

S'utilise pour les appels de DTD et les déclarations d'entités externes

Peut être une adresse absolue :

```
<!DOCTYPE acme SYSTEM "http://www.acme.com/entities/MyDTD.dtd">
```

Peut être une adresse relative :

```
<!DOCTYPE acme SYSTEM "/entities/MyDTD.dtd">
```

```
<!DOCTYPE acme SYSTEM "../../MyDTD.dtd">
```

Les appels en cascade sont résolus relativement à l'objet appelant :

foo.xml

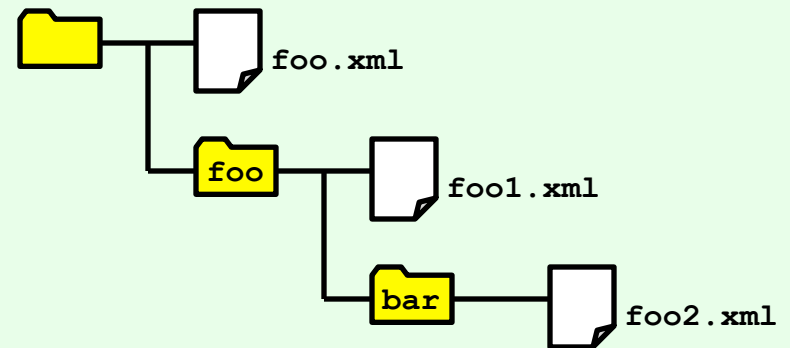
```
<!ENTITY foo-1 SYSTEM "foo/fool1.xml">
```

foo1.xml

```
<!ENTITY foo-2 SYSTEM "bar/foo2.xml">
```

foo2.xml

```
<foo>bar</foo>
```



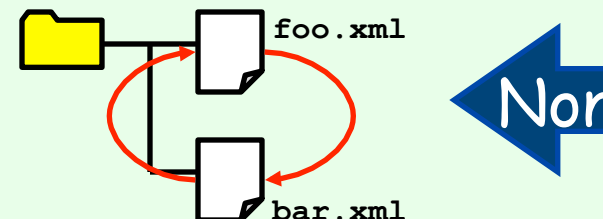
Pas de boucle infinie :

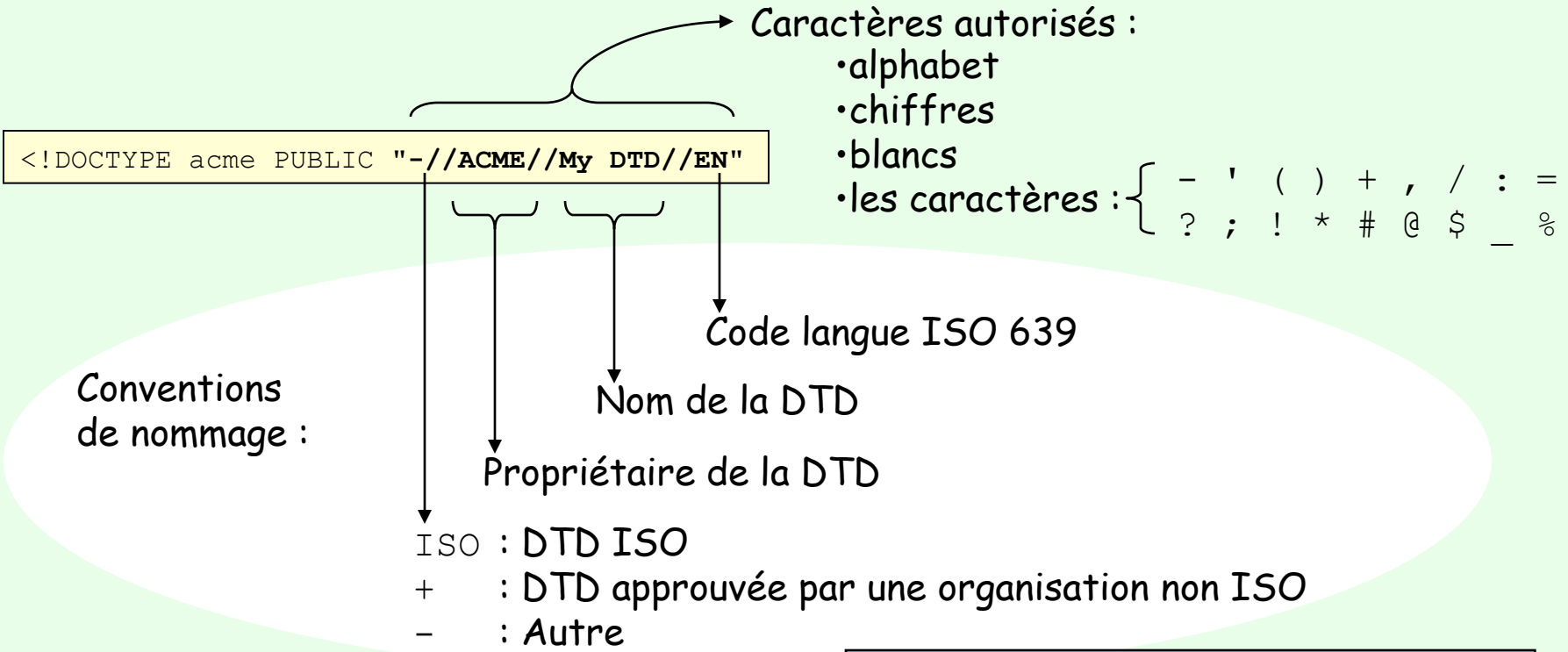
foo.xml

```
<!ENTITY bar SYSTEM "bar.xml">
```

bar.xml

```
<!ENTITY foo SYSTEM "foo.xml">
```





Comment trouver le catalogue ?

- Ça dépend du logiciel qui l'utilise

En utilisant une variable système

```
SGML_CATALOG_FILES=C:\sgml\CATALOG
```

En spécifiant le chemin dans le fichier de configuration du logiciel

En utilisant une classe EntityResolver

- Quels logiciels utilisent un catalogue?
- BALISE
 - EPIC
 - NEAR&FAR
 - SP
 - ...

C'est un résidu de SGML, qu'il est souvent inutile de trimbaler, surtout si l'outil ne le prend pas en compte


```

PUBLIC "ISO 8879-1986//ENTITIES Added Latin 1//EN" "c:\sgml\ISO\ISOLat1.ent"
PUBLIC "ISO 8879-1986//ENTITIES Added Latin 2//EN" "c:\sgml\ISO\ISOLat2.ent"
PUBLIC "ISO 8879-1986//ENTITIES Russian Cyrillic//EN" "c:\sgml\ISO\ISOcyr1.ent"
PUBLIC "ISO 8879-1986//ENTITIES Non-Russian Cyrillic //EN" "c:\sgml\ISO\ISOcyr2.ent"
PUBLIC "ISO 8879-1986//ENTITIES Numeric and Special Graphic//EN" "c:\sgml\ISO\ISONum.ent"
PUBLIC "ISO 8879:1986//ENTITIES Publishing//EN" "c:\sgml\ISO\ISOpub.ent"
PUBLIC "-//TEI//DTD TEI Level 2//EN//2.0" "c:\sgml\tei\tei2.dtd"
PUBLIC '-//TEI P2: 1993//NOTATION WSD for ISO 8859-2//EN' 'C:\SGML\ISO\ISO88592.wsd'
PUBLIC "-//TEI P2: 1993//NOTATION WSD for ISO 8859-1:1986//EN" 'C:\SGML\ISO\iso88591.wsd'
PUBLIC "-//TEI P2 1994//NOTATION WSD for TLG Classical Greek//EN" "c:\CUCUK\teigk2.wsd"

```

Les tendances d'aujourd'hui :

Les parsers permettent de traiter les entités externes par des classes spécifiques (`EntityResolver`)

Il est possible de les utiliser pour conserver une compatibilité avec les "anciens" catalogues publiques (la correspondance doit être effectuée "à la main")

Ils permettent de traiter aussi bien les références `SYSTEM` que `PUBLIC`

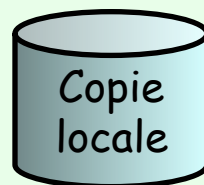
```

<!ENTITY acme PUBLIC "-//ACME//My entity//EN"
"http://www.acme.com/entities/MyEntity.ent">

```

EntityResolver

Utilité



Catalogue
publicID → localURL





```
http://oasis-open.org/committees/entity/spec-2001-08-06.html
```

Java implémentation : <http://www.sun.com/software/xml/developers/resolver/>

Un catalogue pour DocBook :

```
<!DOCTYPE catalog
PUBLIC "-//OASIS//DTD Entity Resolution XML Catalog V1.0//EN"
"http://www.oasis-open.org/committees/entity/release/1.0/catalog.dtd">
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
prefer="public">
  <group xml:base="http://www.oasis-open.org/docbook/xml/4.1.2/">
    <public publicId="-//OASIS//DTD DocBook XML V4.1.2//EN"
      uri="docbookx.dtd"/>
    <public publicId="-//OASIS//ENTITIES DocBook XML Notations V4.1.2//EN"
      uri="dbnotnx.mod"/>
    <public publicId="-//OASIS//ENTITIES DocBook XML Character Entities V4.1.2//EN"
      uri="dbcentx.mod"/>
    <public publicId="-//OASIS//ELEMENTS DocBook XML Information Pool V4.1.2//EN"
      uri="dbpoolx.mod"/>
    <public publicId="-//OASIS//ELEMENTS DocBook XML Document Hierarchy V4.1.2//EN"
      uri="dbhierx.mod"/>
    <public publicId="-//OASIS//ENTITIES DocBook XML Additional General Entities V4.1.2//EN"
      uri="dbgenent.mod"/>
    <public publicId="-//OASIS//DTD DocBook XML CALS Table Model V4.1.2//EN"
      uri="calstblx.dtd"/>
  </group>
  <public publicId="-//OASIS//DTD DocBook MathML Module V1.0//EN"
    uri="http://www.oasis-open.org/docbook/xml/mathml/1.0/dbmathml.dtd"/>
  <nextCatalog catalog="stylesheets.xml"/>
</catalog>
```

Le monde réel

Document sans DTD (sans contrainte de structure)

Que faire avec ce document ?

- Comment l'afficher ?
- Comment accéder à son contenu si les éléments sont inconnus des applications ?

Dans un contexte professionnel, les documents XML seront toujours contraints (DTD, schéma)

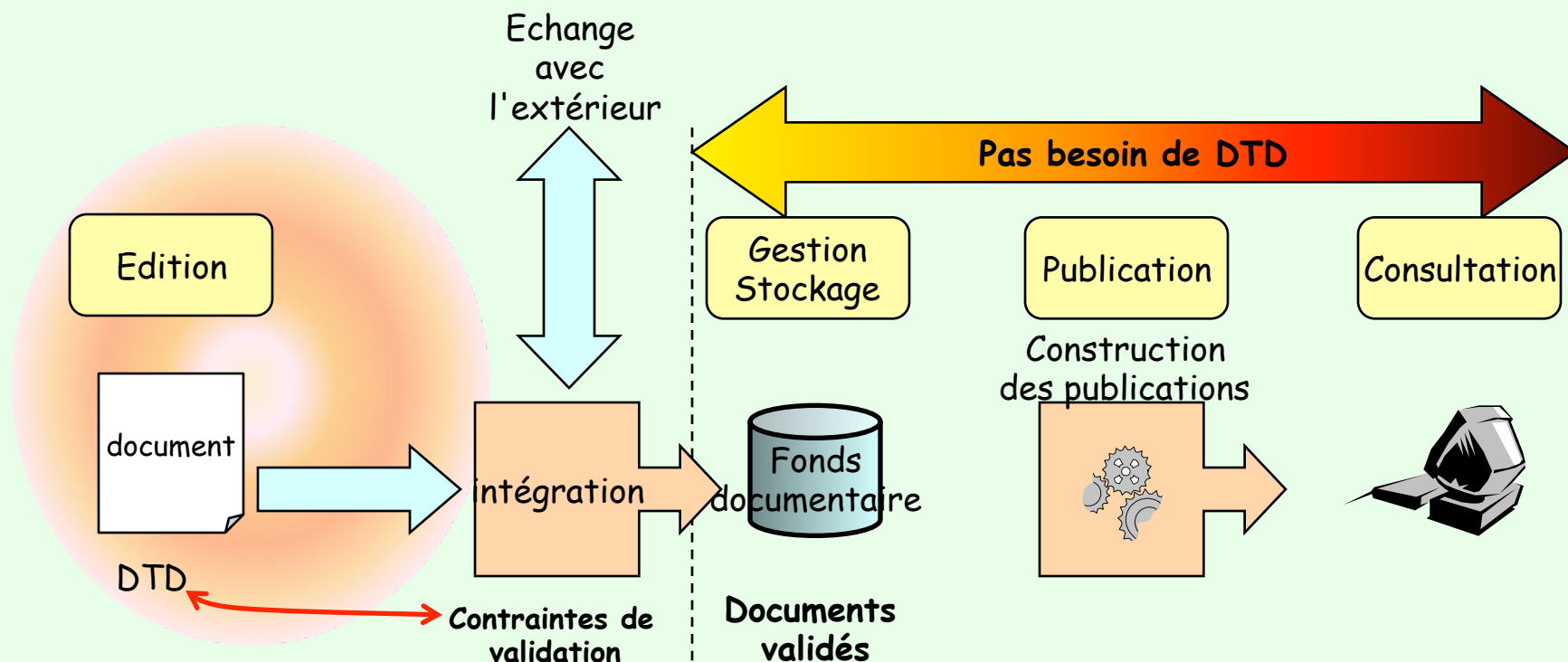
Les contraintes sur les données font partie du standard XML
Autant en profiter pour ne pas surcharger les applications

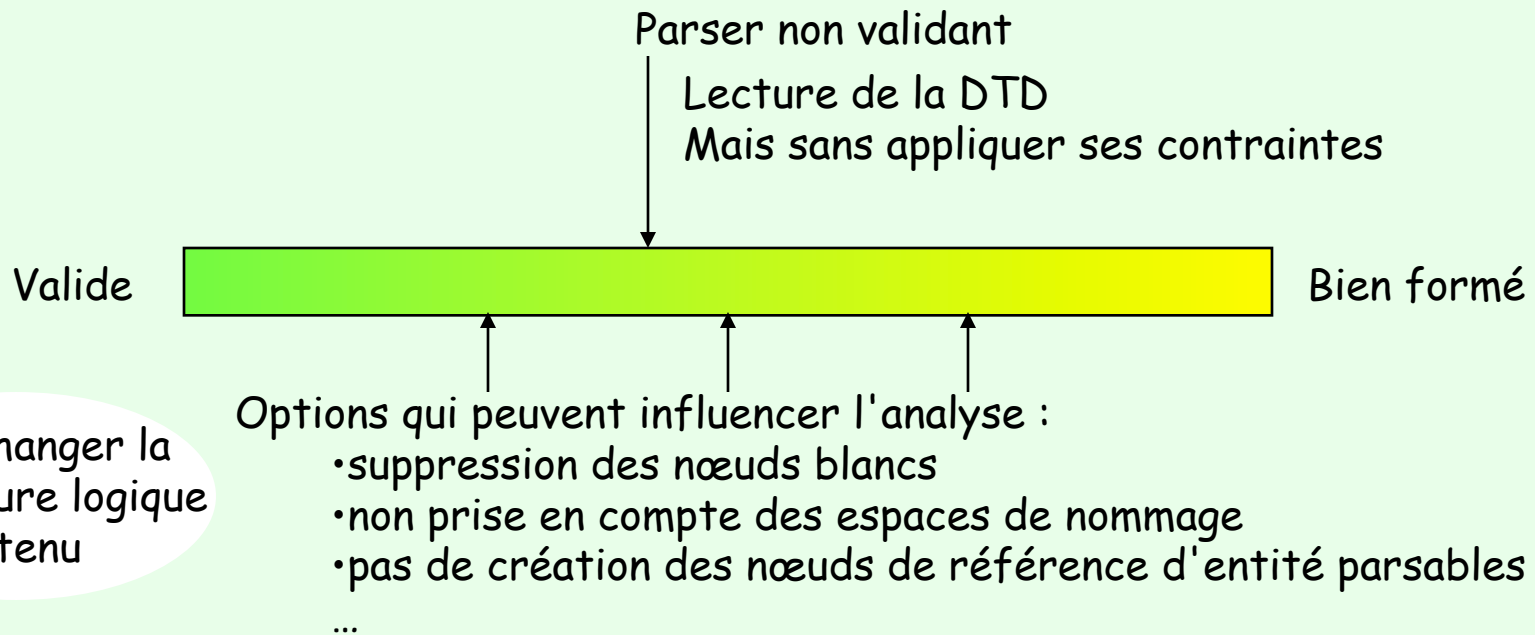
Utilité :

- pour le programmeur et le concepteur (pour XSLT)
- pour les processus qui génèrent des documents XML
- pour lever les contraintes sur certains processus

L'existence d'une DTD permet de servir de support de documentation de la structure de données

...et en cas d'utilisation dans un processus validant de déceler les bugs du programme générateur





Dans la chaîne de publication, on peut se passer de la DTD complète :

Appauvrissement de la DTD

Une deuxième DTD, permettant :

- d'inclure dans le document les entités externes
- d'incorporer la DTD dans le document (en y mettant "juste ce qu'il faut")

```
<?xml version="1.0"?>
<!DOCTYPE foo [
    .../...
]>
<foo>
    .../...
</foo>
```

...et d'utiliser un parseur qui ne valide pas mais qui lit la DTD

Pas de format de données

Pas de contrainte de longueur de champs

Pas de contrainte d'intégrité sur un type d'élément

(les ID \leftrightarrow IDREF ont une portée sur tout le document \forall les éléments)

Syntaxe spécifique

"Verrue" des espaces de nommage

...

Pas de maîtrise des PI, ni des commentaires (qui ne font pas partie de la structure)

Pour valider des contraintes non exprimées (car non exprimable dans les DTD)

\Rightarrow introspection des données (parser dans un parser)

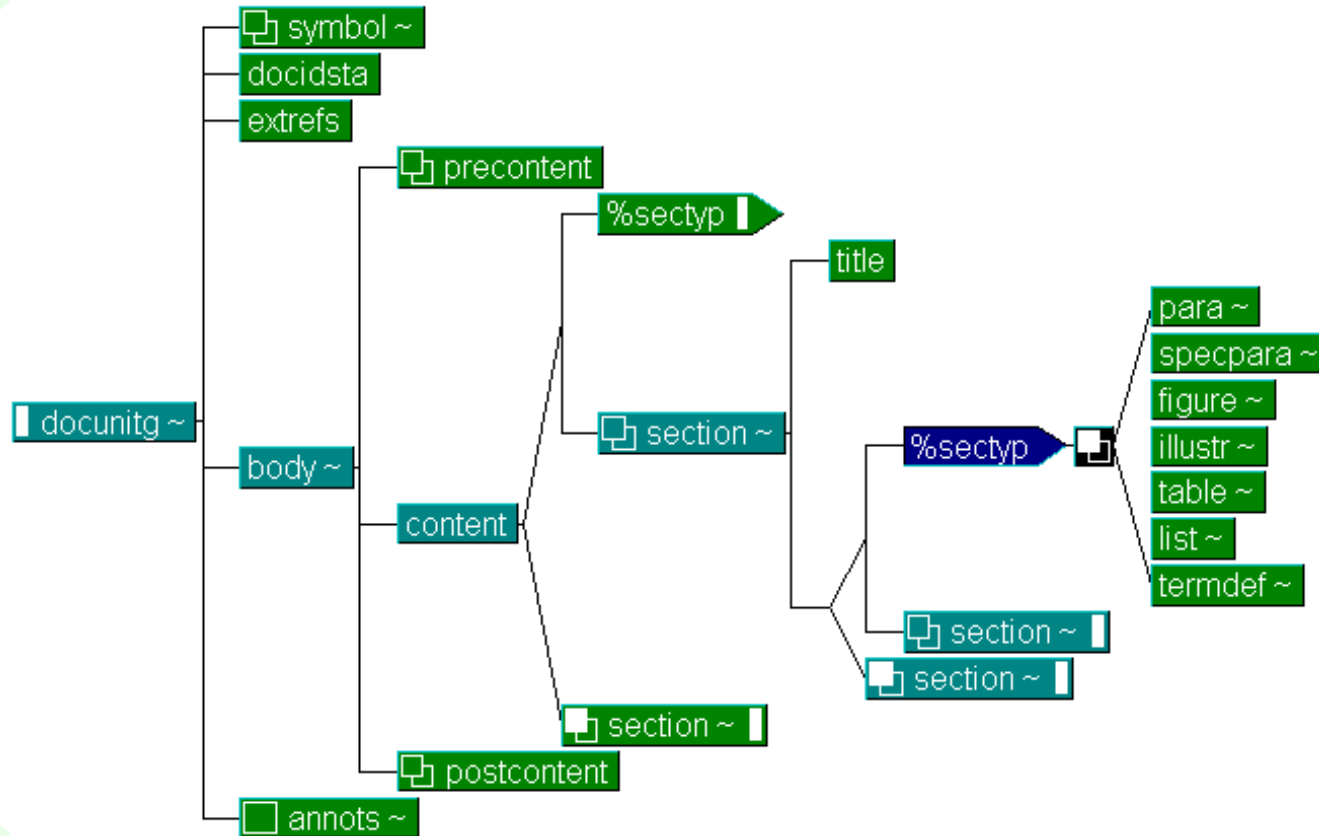
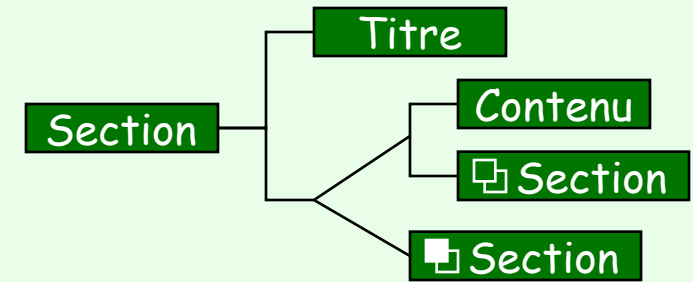
Solutions :

utiliser d'autres schemas

Cependant, toutes les technologies de schémas ont leur limitations

Représentation graphique des DTD

- XML authority
- Near&Far designer



Dans mon entreprise,
je crée mes DTD pour
mes propres besoins.

Utiliser les standards métiers existants :

Ma DTD : {

- Un peu de RDF pour les Meta-Données
- Un peu de XHTML pour les données documentaires
- Un peu de SVG pour insérer des graphiques
- Un peu de MathML pour insérer des formules mathématiques
- Un peu de XLink pour décrire des liens complexes
- Un peu de mes balises issues de mon métier
- ...

→ Instrumentation des DTD

- construire sa DTD en piochant dans les standards existants

Utiliser les applications XML liées au standard pour la modélisation et les traitements :

- WXS : conception
- XSLT : publication cross-media
- DOM : traitement du contenu (BdD...)
- XCL : intégration
- ...

Entités externes binaires

A utiliser seulement si vos outils les supportent

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE zoo [
  <!NOTATION jpg SYSTEM "images/jpeg">
  <!ELEMENT dauphin (nom, taille, poids)>
  <!ATTLIST dauphin
    id          ID #REQUIRED
    photo       ENTITY #IMPLIED>
  <!ENTITY flipper.jpg SYSTEM "flipper.jpg" NDATA jpg>
  .../...
]>
<zoo>
  <aquarium>
    <dauphin id="jhgtr13" photo="flipper.jpg">
      <nom>Flipper</nom>
      .../...
    </dauphin>
  </aquarium>
</zoo>
```

- Marque l'appartenance au document
- Permet à certains éditeurs l'affichage

La redondance `flipper.jpg` permet de dissocier le document de sa DTD pour des traitements ayant lieu après la validation

```
<dauphin id="jhgtr13" photo="flipper.jpg">
```

- Lors de l'édition, la valeur de l'attribut `photo` référence l'entité externe binaire
- Lors de la publication, la valeur de l'attribut `photo` référence le nom du fichier

Factoriser les DTD en utilisant les entités générales

Dans un document XML :

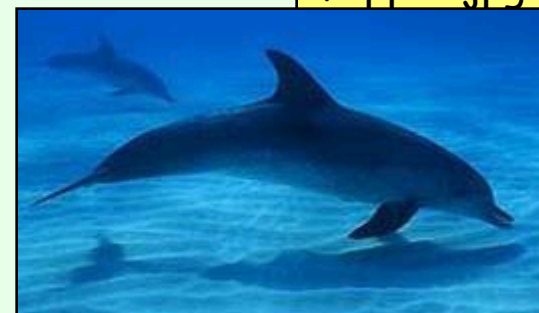
- définir l'utilisation des parties optionnelles (INCLUDE ou IGNORE)
 - appeler les modules généraux, comme les notations
 - définir les entités qui "font partie" du document (ça ne sert à rien ?)
(→ penser aux cas où ça peut servir)
- C'est un moyen simple qui permet à des traitements de prendre en compte les médias contenus dans un document

notations.ent

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!NOTATION jpeg SYSTEM "images/jpeg">
<!NOTATION gif SYSTEM "images/gif">
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE zoo SYSTEM "zoo.dtd" [
  <!ENTITY % notations SYSTEM "notations.ent">
%notations;
  <!ENTITY % svg_structure SYSTEM "svg.dtd" >
%svg_structure;
  <!ENTITY flipper.jpg SYSTEM "flipper.jpg" NDATA jpeg>
  <!ENTITY oum SYSTEM "oum.xml">
]>
<zoo>
  <aquarium>
    <dauphin id="jhgtr13" photo="flipper.jpg">
      <nom>Flipper</nom>
      .../...
    </dauphin>
  </aquarium>
</zoo>
```

flipper.jpg

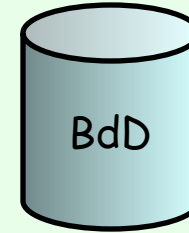


Edition à partir d'un éditeur XML

Exemple : construction d'un rapport

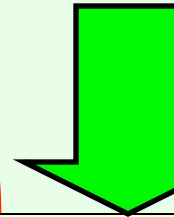
```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE rapport SYSTEM "rapport.dtd" [
  <!ENTITY ventes SYSTEM "ventes.xml">
]>
<rapport>
  &ventes;
  <analyse>
    <p>Cette année, la croissance
    est au rendez-vous,
    .../...
    </p>
  </analyse>
</rapport>
```

Système d'information



BdD

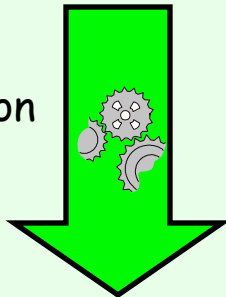
Génération automatique du fichier



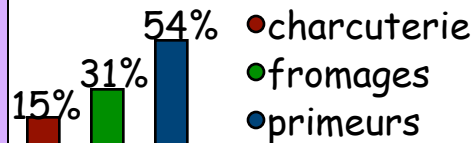
ventes.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<ventes>
  <charcuterie>23</charcuterie>
  <fromages>45</fromages>
  <primeurs>78</primeurs>
</ventes>
```

Publication



Web Explorer



Cette année, la croissance est au rendez-vous ...

W3C

XML

Schema

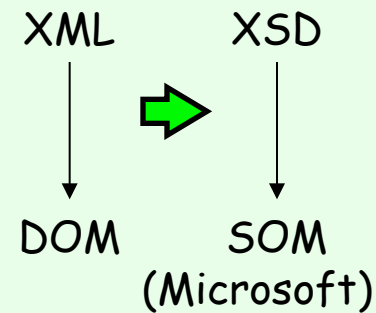
DTD

- Spécifie un modèle de contenu

XML Schema definition language (XSD)

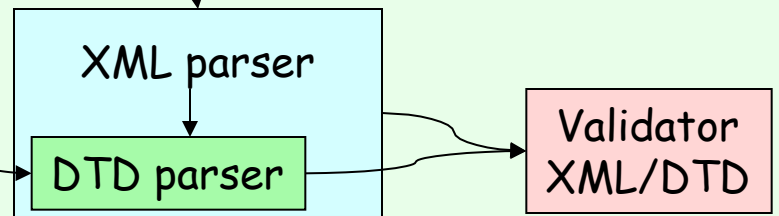
WXS ou XSD

- Syntaxe XML
- Supporte les format de données basiques
- Permet l'extension des types de données



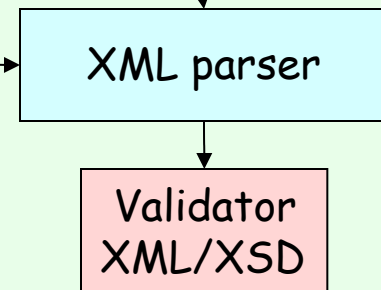
```

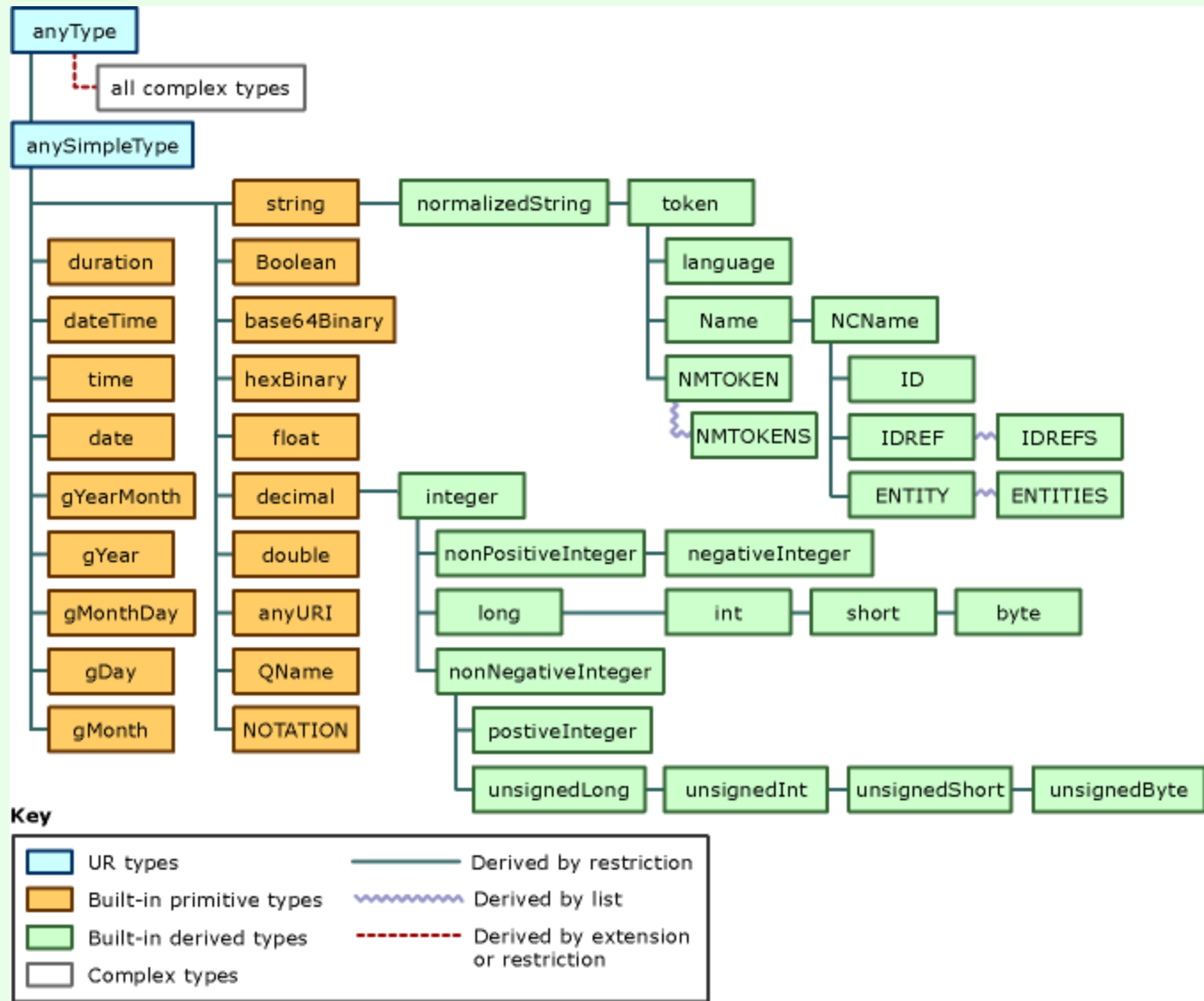
<?xml version="1.0"?>
<!DOCTYPE document SYSTEM "document.dtd">
<document>
    .../...
</document>
    
```



```

<?xml version="1.0"?>
<document xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://document.org document.xsd">
    .../...
</document>
    
```





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

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

```
<!ELEMENT name (#PCDATA)>  
with attribute-list declarations
```

```
<xsd:element name="name">  
  <xsd:complexType>  
    <xsd:simpleContent>  
      <xsd:extension base="xsd:string">  
        <xsd:attributeGroup ref="foo"/>  
      </xsd:extension>  
    </xsd:simpleContent>  
  </xsd:complexType>  
</xsd:element>
```

```
<!ELEMENT name (#PCDATA | foo1 | foo2 | ...) *>  
with attribute-list declarations
```

```
<xsd:element name="name">  
  <xsd:complexType mixed="true">  
    <xsd:choice minOccurs="0" maxOccurs="unbounded">  
      <xsd:element ref="foo1"/>  
      <xsd:element ref="foo2"/>  
      ...  
    </xsd:choice>  
    <xsd:attributeGroup ref="foo"/>  
  </xsd:complexType>  
</xsd:element>
```

```
<!ELEMENT name (element-content-model)>  
with attribute-list declarations
```

```
<xsd:element name="name">  
  <xsd:complexType>  
    element-content-model  
    <xsd:attributeGroup ref="foo"/>  
  </xsd:complexType>  
</xsd:element>
```

Référence des autres éléments dans le modèle de contenu

```
<xsd:element ref="name"/>
```

, dans le modèle de contenu

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

| dans le modèle de contenu

```
<xsd:choice>...</xsd:choice>
```

* dans le modèle de contenu

```
minOccurs="0" maxOccurs="unbounded"
```

+ dans le modèle de contenu

```
minOccurs="1" maxOccurs="unbounded"
```

? dans le modèle de contenu

```
minOccurs="0" maxOccurs="1"
```



```
<!ATTLIST name  
  attribute-declarations>
```

attribute declaration

#IMPLIED

#REQUIRED

CDATA

```
<xsd:attributeGroup name="name">  
  attribute-declarations  
</xsd:attributeGroup>
```

```
<xsd:attribute name="..." ...>  
  ...  
</xsd:attribute>
```

use="optional" (**default**)

use="required"

type="xsd:string"

```
<!-- ... -->
```

```
<!ENTITY % param-ent-name  
  attribute-declarations>
```

```
<!ENTITY % param-ent-name  
  element-content-model>
```

```
%param-ent-name;
```

```
%param-ent-name;
```

```
<!ENTITY name ...>
```

Déclarations d'entités parsées ou non

```
<xsd:annotation>  
  <xsd:documentation>  
    ...  
  </xsd:documentation>  
</xsd:annotation>
```

```
<xsd:attributeGroup name="param-ent-name">  
  attribute-declarations  
</xsd:attributeGroup>
```

```
<xsd:modelGroup name="param-ent-name">  
  element-content-model  
</xsd:modelGroup>
```

```
<xsd:attributeGroup ref="param-ent-name" />
```

```
<xsd:modelGroup ref="param-ent-name" />
```

Non applicable

Relax NG

(Technical Report ISO/IEC TR 22250-1:2001 de l'ISO/IEC JTC1)

Relax Core (James Clark)

TREX

Relax NG

(OASIS, puis ISO)

Un super bouquin complet et en ligne :

<http://books.xmlschemata.org/relaxng/>

<http://www.oasis-open.org/committees/relax-ng/>

<http://www.relaxng.org/>

Forme XML

```
<element name="author">
  <attribute name="id"/>
  <ref name="name-element"/>
  <optional>
    <element name="born">
      <text/>
    </element>
  </optional>
  <optional>
    <element name="died">
      <text/>
    </element>
  </optional>
</element>
```

Forme Compacte

```
element author {
  attribute id { text },
  name-element,
  element born { text }?,
  element died { text }?
}
```

Permet d'exprimer à peu près les mêmes choses que les DTD :

- dans une forme XML
- avec beaucoup plus de puissance d'expression
- en utilisant le typage de données

```
<!ELEMENT name (#PCDATA)>
with attribute-list declarations
```

```
<define name="name">
  <element name="name">
    attribute-declarations
    <text/>
  </define>
```

```
<!ELEMENT name (#PCDATA | foo1 | foo2 | ...) *>
with attribute-list declarations
```

```
<define name="name">
  <element name="name">
    attribute-declarations
    <mixed>
      <zeroOrMore>
        <choice>
          <ref name="foo1"/>
          <ref name="foo2"/>
          ...
        </choice>
      </zeroOrMore>
    </mixed>
  </define>
```

```
<!ELEMENT name (element-content-model)>
with attribute-list declarations
```

```
<define name="name">
  <element name="name">
    attribute-declarations
    element-content-model
  </define>
```

Référence des autres éléments dans le modèle de contenu

```
<ref name="name"/>
```

, dans le modèle de contenu

```
<group>...</group>
```

| dans le modèle de contenu

```
<choice>...</choice>
```

* dans le modèle de contenu

```
<zeroOrMore>...</zeroOrMore>
```

+ dans le modèle de contenu

```
<oneOrMore>...</oneOrMore>
```

? dans le modèle de contenu

```
<optional>...</optional>
```

attribute declaration

```
<attribute name="...">  
  ...  
</attribute>
```

#IMPLIED

```
<optional>...</optional>
```

CDATA

```
<text/>
```

```
<!-- ... -->
```

```
<a:documentation>
...
</a:documentation>
```

```
<!ENTITY % param-ent-name
attribute-declarations>
```

```
<define name="param-ent-name">
attribute-declarations
</define>
```

```
<!ENTITY % param-ent-name
element-content-model>
```

```
<define name="param-ent-name">
element-content-model
</define >
```

```
%param-ent-name;
```

```
<ref name="param-ent-name" />
```

```
%param-ent-name;
```

```
<ref name="param-ent-name" />
```

```
<!ENTITY name ...>
```

Non applicable

Déclarations d'entités parsées ou non

Relax NG définit 2 types de données : "String" et "token", mais offre la possibilité d'utiliser des bibliothèques externes.

En particulier, Relax NG peut utiliser les types de données et les facettes des schémas XML du W3C

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<element name="itemList"
  xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <zeroOrMore>
    <element name="item">
      <attribute name="number"/>
      <element name="name">
        <data type="token">
          <a:documentation>Jusqu'à 5 caractères</a:documentation>
          <param name="maxLength">5</param>
        </data>
      </element>
      <element name="quantity">
        <data type="short">
          <a:documentation>Nombres de 0 à 20</a:documentation>
          <param name="maxInclusive">20</param>
          <param name="minInclusive">0</param>
        </data>
      </element>
    </element>
  </zeroOrMore>
</element>
```

PSVI et validation

L'augmentation consiste à compléter le modèle de données grâce à des informations contenues dans le schéma

Bien qu'avantageuse, cette technique pose des problèmes :

- information manquante quand on n'a pas accès au schéma
- technique non normalisée, sauf pour les DTD (DOM)

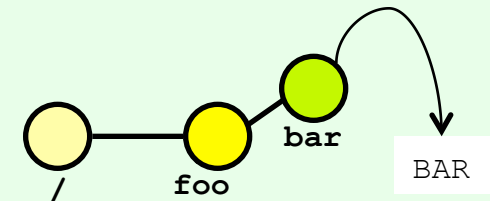
Mais offre aussi des avantages :

- typage des données fourni par le schéma (avec WXS)
- data-binding grâce au typage

Augmentation avec les DTD

Les valeurs par défaut des attributs fournis par la DTD sont fixées dans le modèle de données

```
<?xml encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ATTLIST foo bar CDATA #FIXED "BAR">
]>
<foo/>
```



→ Le schéma "augmente" les informations contenues dans le document

PSVI = data augmentation

Post Schema Validation Infoset

PSVI est un mécanisme d'augmentation propre à WXS

Problème :

- une API très limitée pour accéder aux données supplémentaires
- obligation d'avoir recours à des API propriétaires

Exemple avec Xerces :

```
//dbf is JAXP DocumentBuilderFactory
// all of the following features must be set:
dbf.setNamespaceAware(true);
dbf.setValidating(true);
dbf.setAttribute("http://apache.org/xml/features/validation/schema", Boolean.TRUE);
// you also must specify Xerces PSVI DOM implementation
// "org.apache.xerces.dom.PSVIDocumentImpl"
dbf.setAttribute("http://apache.org/xml/properties/dom/document-class-name",
    "org.apache.xerces.dom.PSVIDocumentImpl");

...
Document doc = db.parse(args[0]);
if (doc.getDocumentElement().isSupported("psvi", "1.0")){
    ElementPSVI psviElem = (ElementPSVI)doc.getDocumentElement();
    XSElementDeclaration decl = psviElem.getElementDeclaration();
    ...
}
```

Utilisé dans XSLT 2.0

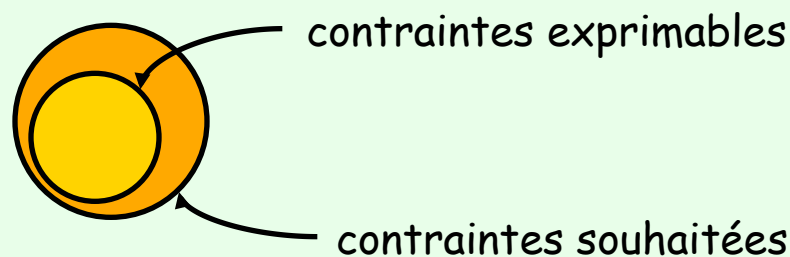
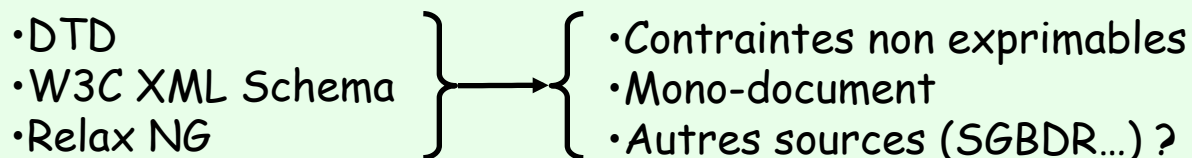
Par exemple :

- un tri par date devient possible dans XSLT 2.0 sur une donnée de ce type

- Celui qu'on connaît
- Celui qui est supporté par les outils qu'on utilise
- Celui qui correspond à ses exigences en termes d'expressions de contrainte

Incomplétude de la validation par schéma

On ne peut pas tout valider avec un schéma



Les schémas laissent toujours plus de liberté que nécessaire

Complément : validation par programmation

- Permet seulement de valider
- Ne permet pas de savoir ce qui est autorisé dans un contexte donné

<http://www.schematron.com/>

The Schematron Assertion Language

Devenu une norme ISO

Basé sur des
règles

```
<?xml version="1.0" encoding="iso-8859-1"?>
<sch:schema xmlns:sch="http://www.ascc.net/xml/schematron">
  <sch:title>Example Schematron Schema</sch:title>
  <sch:pattern>
    <sch:rule context="dog">
      <sch:assert test="count(ear) = 2">A 'dog' element should
          contain two 'ear' elements.</sch:assert>
      <sch:report test="bone">This dog has a bone.</sch:report>
    </sch:rule>
  </sch:pattern>
</sch:schema>
```

Validation partielle

```
<phase id="basicValidation">
  <active pattern="text" />
  <active pattern="tables" />
  <active pattern="attributePresence" />
</phase>
<phase id="fullValidation">
  <active pattern="metadata" />
  <active pattern="text" />
  <active pattern="tables" />
  <active pattern="attributePresence" />
  <active pattern="attributeValueChecks" />
</phase>
```

- Permet seulement de valider
- Ne permet pas de savoir ce qui est autorisé dans un contexte donné