

Fondamentaux

XML

L'univers XML

XML : un métalangage

Le document XML

XML et ses représentations

Syntaxe

Le balisage

Les attributs

Commentaires, PIs

Prologue

Entités

Sections CDATA

Noms XML

Normalisation des blancs

Traitement des blancs

Documents bien formés

Composition des documents

Parseurs

Limitations XML

Background

Edition

Standards connexes

Applications XML

Utiliser XML

Conception

Structuration des données

Règles pragmatiques

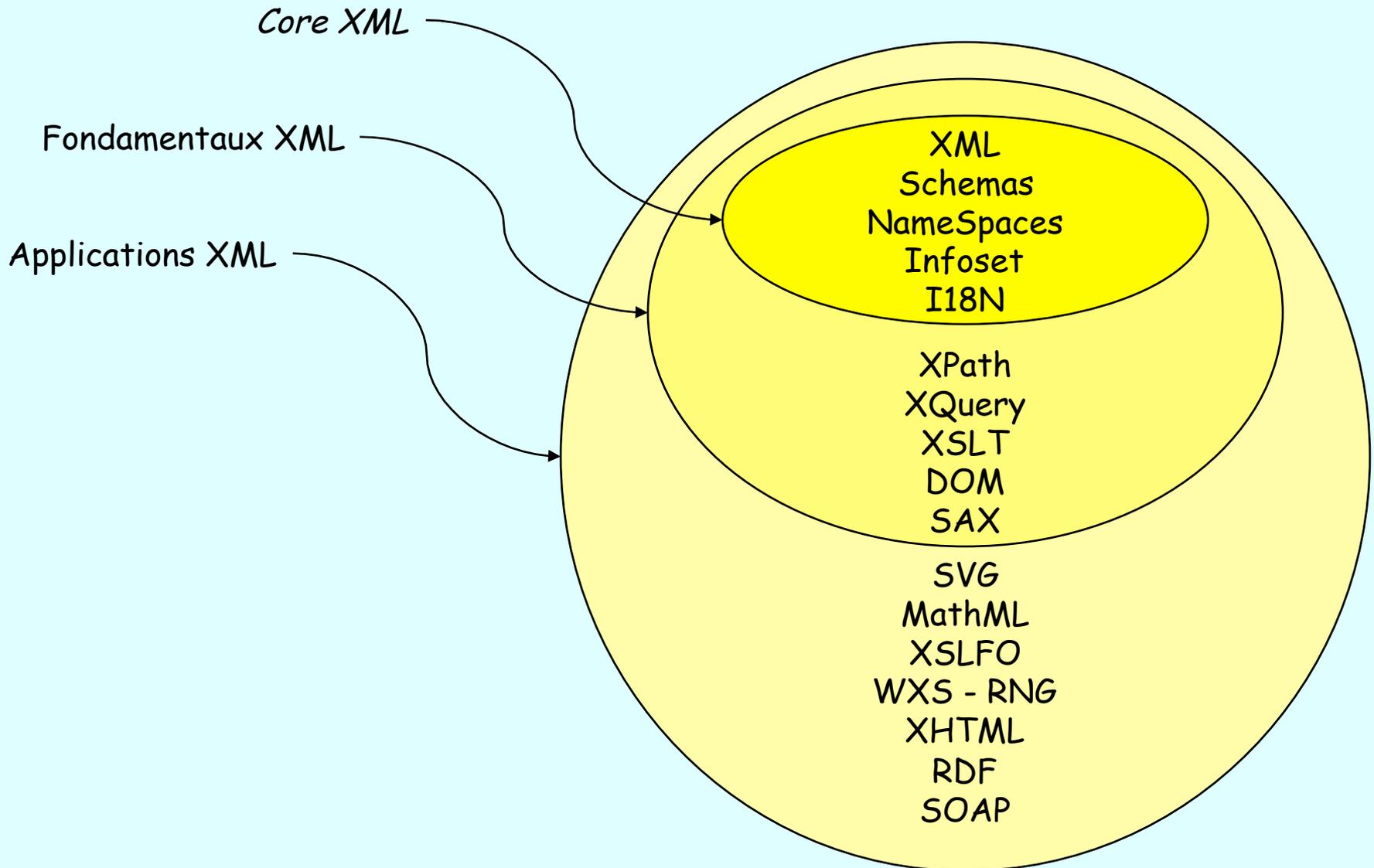
Shorthand properties

Atomicité des données

Cas des dates

Conception des structures de données

Typologie des structures



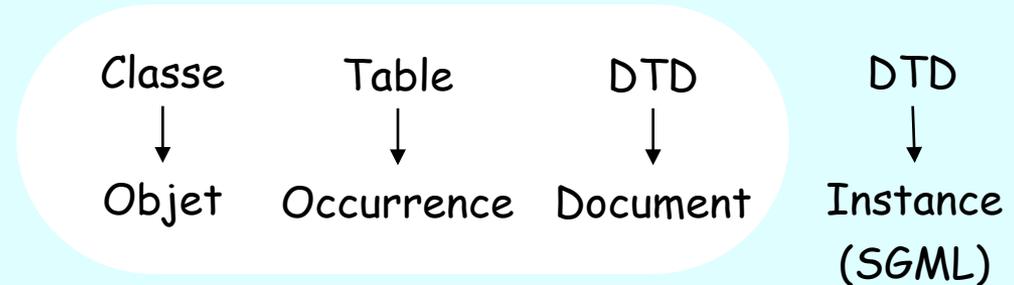
Métalangage : langage qui permet de concevoir d'autres langages

Langage de balisage : langage de description de document
(n'est pas un langage de programmation)

Description d'une classe de document : DTD (Document Type Definition)

La DTD décrit l'ensemble des règles qui définissent la structure d'une classe de documents XML :

- arborescence des éléments
- ordre des éléments
- fréquence d'apparition
- attributs des balises
- .../...



Concept fondamental

Un document XML peut-être soit :

• **bien formé** : respect de la syntaxe XML

soit :

• **valide** : respect de la syntaxe XML

ET

conforme vis à vis d'une DTD désignée

Document : terme à prendre au sens large

Le document n'est pas nécessairement un fichier

Il peut s'agir :

- d'informations stockées en BdDR
- d'un flux de données
- de n'importe quelle autre source de données

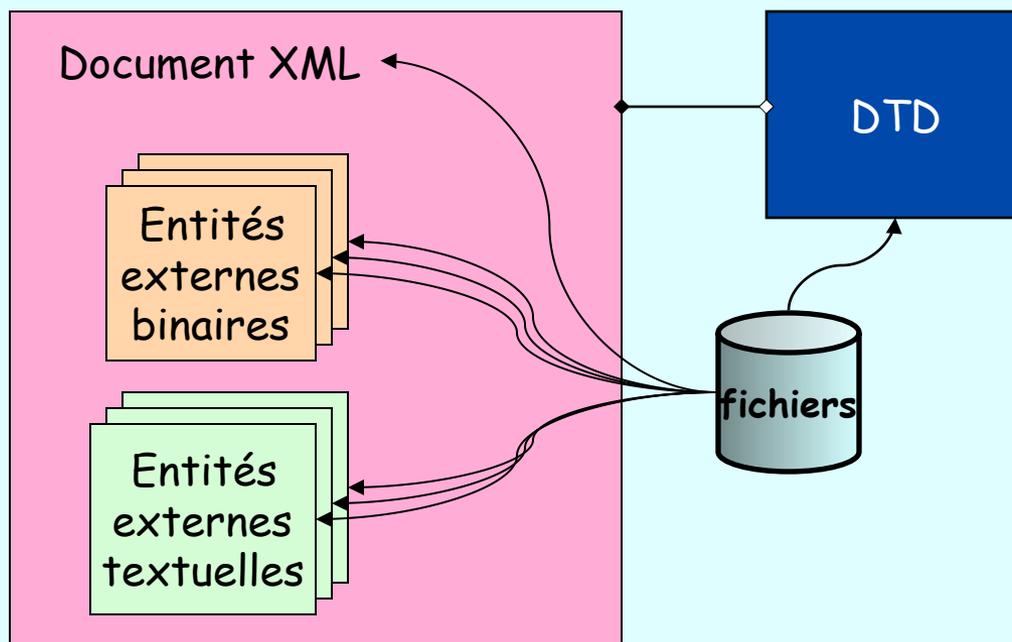
Le terme "fichier", même s'il est impropre, est la plupart du temps utilisé.

Le document est souvent **composite** (images, vidéo, son...),

→ constitué de plusieurs "fichiers"

Document composé de plusieurs fichiers :

- fichier XML principal
- autres fragments XML
- ressources binaires



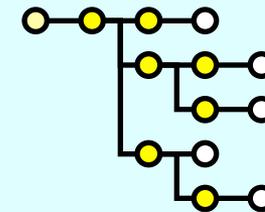
Structure physique → forme sérialisée

Séquence de caractères "à plat"
 Spécifie l'encodage caractère
 Spécifie le découpage en fichiers
 Peut contenir des caractères non significatifs
 (indentations)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Course SYSTEM "Course.dtd">
<Course>
  <Title>XML course</Title>
  <Author>  <LastName>Poulard</LastName>
             <FirstName>Philippe</FirstName>
  </Author>
  <Description>A course about <b>XML</b>
               core technologies.
  </Description>
</Course>
```

Modèle logique

Représentation abstraite du modèle physique
 Structure de données arborescente
 Plusieurs APIs
 Est délivré aux applications (processeur XSLT)



→ XML infoset

<http://www.w3.org/TR/2001/REC-xml-infoset-20011024>

Forme canonique → signature

Un même modèle logique peut être représenté par plusieurs structures physiques

→ la **forme canonique** du document permet de comparer 2 documents

<http://www.w3.org/TR/2001/REC-xml-c14n-20010315>

On y trouve :

Du contenu

```
Le corbeau et le renard
```

Des éléments (dont la forme concrète est représentée par des balises)

```
<Fable>
```

Des paramètres (ou caractéristiques) applicables à ces éléments, les **attributs**

```
<Fable type="vers">
```

Des instructions de traitement destinées à des applications spécifiques

```
<?xml-stylesheet href="mystyle.css" type="text/css"?>
```

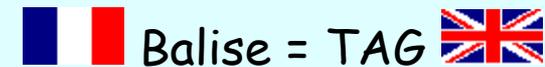
Des commentaires

```
<!-- était-ce un camembert ou un roquefort ? -->
```

Eventuellement, une référence externe à la structure (DTD)

```
<!DOCTYPE Fable SYSTEM "fable.dtd">
```

Un élément a un **nom** : `monElément`



la casse n'est pas ignorée :

`monélément`, `MONELEMENT`, et `MonElément` sont des éléments différents

pour utiliser un élément, on dispose :

d'une **balise ouvrante** :

`<monElément>`

et d'une **balise fermante** :

`</monElément>`

le texte compris entre une balise ouvrante et une balise fermante (le **contenu**) est supposé être doté des propriétés structurelles de l'élément :

`<monElément>texte doté des propriétés de l'élément</monElément>`



Il faut toujours fermer une balise qui a été ouverte.

Les balises peuvent s'imbriquer les unes dans les autres :

`texte1 <i> texte2 texte3 </i>`

Non

`texte1 <i> texte2 </i> texte3 `

Oui

Attribut : paramètre appliqué à un élément

- un attribut a un **nom** et une **valeur** :

```
monattribut="lavaleur"
```

Simple quotes '
ou
double quotes "

- un attribut s'utilise avec un élément :

mabalise s'utilise avec l'attribut monattribut

- pour utiliser un attribut avec un élément, on l'écrit dans la balise ouvrante :

```
<monElément monattribut="lavaleur">
```

- une balise peut avoir **plusieurs** attributs différents (l'ordre n'a pas d'importance) :

```
<monElément monattribut="lavaleur" monautreattr="val">
```



les blancs contenus dans la valeur ne sont pas significatifs (→ *normalization*)

Certaines balises n'ont pas de contenu :
elles peuvent se suffire à elles-mêmes

ou "fonctionner" grâce à leurs attributs, comme la balise `img` en HTML
(qui permet d'insérer un fichier contenant une illustration).

En XML, on écrirait :

```
<image source="uneimage.gif"></image>
```

ou la forme abrégée :

```
<image source="uneimage.gif"/>
```

} Structures
logiques
équivalentes

Commentaire : information ignorable
commence par `<!--` et termine par `-->`

Un commentaire ne peut pas contenir `--`

```
<!--Ceci est un commentaire-->
```

PI (Processing Instruction) : instruction de traitement
Destiné à une application spécifique
commence par `<?` et termine par `?>`

S'écrivent hors des balises ouvrantes et fermantes

```
<?robots index="yes" follow="no"?>
```

Diagramme de l'exemple de PI ci-dessus :

- Le mot `robots` est désigné par une accolade et étiqueté "cible".
- Les attributs `index="yes"` et `follow="no"` sont désignés par une accolade et étiquetés "données".
- Les attributs `index="yes"` et `follow="no"` sont désignés par une accolade et étiquetés "Pseudo-attributs".

Un processeur XML ne traite pas ces instructions mais les transmet à l'application cible, qui peut les ignorer.

Les PI qui commencent par `xml` ont un usage réservé par le standard.

Déclaration XML : "PI particulière" placée en début de document

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

`standalone="yes"`

Autonomie du document :
(peu utile)

- Documents sans DTD
- Documents avec DTD internes
- Documents dont les DTD ne les modifient pas

Le **prologue** d'un document XML contient tout ce qui précède l'**élément racine**

On peut y trouver :

- la déclaration XML ← toujours sur la première ligne
- des instructions de traitement
- des commentaires
- une clause DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="bar.xsl"?>
<!DOCTYPE foo SYSTEM "foo.dtd">
<!--mise à jour du 2000-11-01-->
```

Prologue

<foo>

.../...

</foo>

Balise fermante de l'élément racine

Balise ouvrante de l'élément racine



Après la balise fermante de l'élément racine, on peut trouver des commentaires, des instructions de traitement, et des blancs

Il est fortement recommandé de doter ses documents d'une déclaration XML



Rien ne doit précéder une déclaration XML (pas même des blancs), sauf, éventuellement, des caractères BOM (*byte order mark*) pour les documents encodés sur 16-bits. Les caractères BOM sont considérés comme ne faisant pas partie du document.

Les appels d'entité permettent :

- d'échapper les caractères réservés

Si a<b & a>c alors...

Rendu : Si a<b & a>c alors...

- d'inclure des caractères qui ne font pas partie du jeu utilisé

man<#339>uvre

Rendu : manœuvre

- d'inclure des morceaux de documents XML

©right;

Le texte de remplacement est défini dans la DTD

- d'insérer des références à des fichiers binaires (images...)

<image photo="flipper">

Cas particulier : l'entité `flipper` n'est pas encadrée par `&` et ;
C'est la DTD qui indique que l'attribut `photo` fait référence à une entité binaire

Entités prédéfinies :

Nom d'entité	Appel	Caractère
lt	<	<
gt	>	>
amp	&	&
quot	"	"
apos	'	'

Utilisation des entités :

- déclaration dans la DTD → sauf...
- **appel** dans le contenu des éléments et des attributs

les entités caractères ←

Plutôt que d'échapper les 5 entités prédéfinies dans un long texte, il est parfois utile d'encadrer l'ensemble dans une section CDATA (*character data*).

```
<exemple>
Ce qui suit est un exemple de document XML :
<![CDATA[
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Fable SYSTEM "Fable.dtd">
<Fable>
  <Titre>Le corbeau et le renard</Titre>
  <Auteur>
    <Nom>De la Fontaine</Nom>
    <Prénom>Jean</Prénom>
  </Auteur>
</Fable>
]]>
</exemple>
```

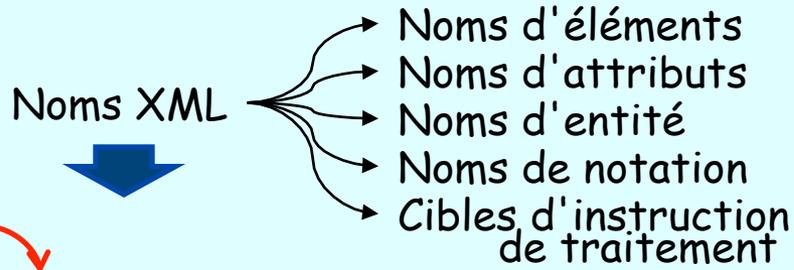
```
<exemple>
Ce qui suit est un exemple de document XML :
<?xml version="1.0"
      encoding="ISO-8859-1"?>
  <![DOCTYPE Fable SYSTEM "Fable.dtd"&gt;
  <![Fable&gt;
    &lt;Titre&gt;Le corbeau et le renard
    &lt;/Titre&gt;
    &lt;Auteur&gt;
      &lt;Nom&gt;De la Fontaine&lt;/Nom&gt;
      &lt;Prénom&gt;Jean&lt;/Prénom&gt;
    &lt;/Auteur&gt;
  &lt;/Fable&gt;
</exemple>
```



Ceci est le
contenu textuel
de l'élément
exemple

```
Ce qui suit est un exemple de document XML :
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Fable SYSTEM "Fable.dtd">
<Fable>
  <Titre>Le corbeau et le renard</Titre>
  <Auteur>
    <Nom>De la Fontaine</Nom>
    <Prénom>Jean</Prénom>
  </Auteur>
</Fable>
```

Une section CDATA ne peut apparaître que dans le contenu d'un élément



Remarque sur les caractères interdits dans les unités lexicales nominales :

Blancs : { retours chariots, espaces, espaces insécables, tabulations

Les noms d'élément et d'attribut ne peuvent pas commencer par `xml` (casse indifférente) : usage réservé par le standard

Chiffres : 0-9

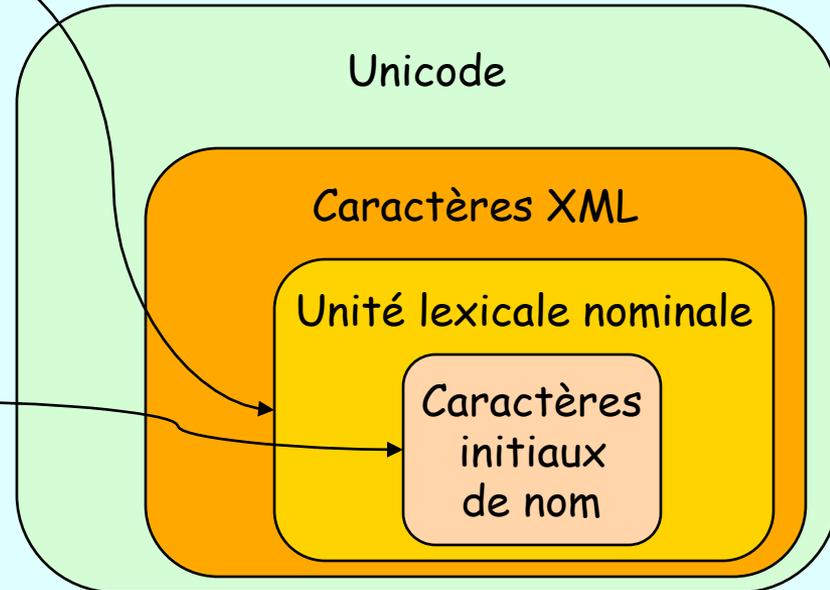
Caractères de ponctuation :
- (tiret)
. (point)

Caractère spécial :
: (deux points, namespace)

Alphabet accentué : a-z A-Z

Caractère de ponctuation :
_ (souligné)

Caractères utilisés dans les langues non romanes



Traitement du blanc (espaces, tabulations, interlignes)

Processeur XML → transmission des blancs à l'application
Processeur XML validateur → transmission des blancs à l'application

⊕

indique si les blancs apparaissent dans du contenu élémentaire pur

Windows : CRLF
MacOS : CR
Unix : LF

Normalisation des fins de ligne

#xD#xA	}	#xA		
#xD				
[#xA	^M	LF	LINE FEED
	#xD	^J	CR	CARRIAGE RETURN

Ne contient que des sous-éléments (pas de données textuelles)

```

<dauphin>
  <nom>Flipper</nom>
  <sexe>M</sexe>
</dauphin>

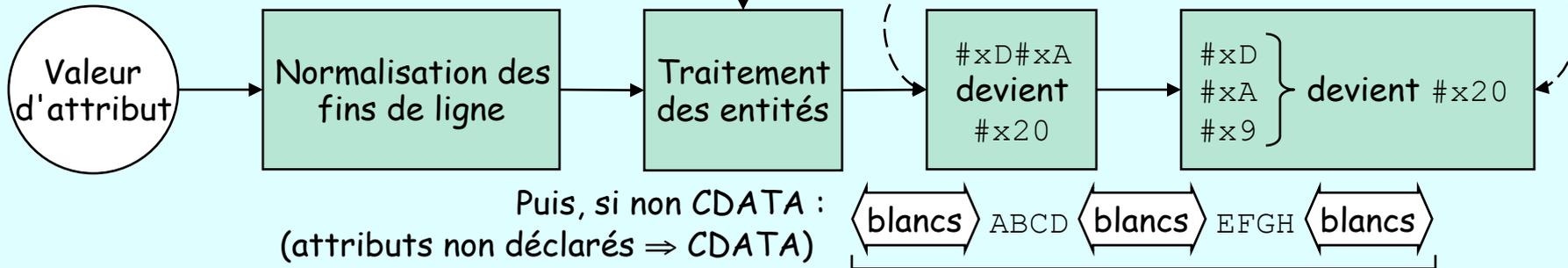
```

Normalisation des attributs

Appel d'entité

Appel de caractère

(sauf pour les appels de caractère)



Puis, si non CDATA : (attributs non déclarés ⇒ CDATA)

```

<foo bar="abc def&#xA;ghi
  jkl mno"/>

```

```

abc def
ghi jkl mno

```

devient #x20 ABCD #x20 EFGH #x20

Le parser transmet les espaces blancs à l'application



L'application détermine quels espaces blancs sont significatifs

Pour forcer l'application à conserver les espaces blancs, on utilise l'attribut réservé :

```
xml:space="preserve"
```

dans un document XML

Les sous éléments héritent de la fonctionnalité de l'attribut

Pour arrêter la propagation aux sous-éléments, utiliser

```
xml:space="default"
```

Doit être déclaré dans la DTD (pour les parsers validants)

```
<!ATTLIST unElement
    xml:space (default |preserve) "preserve">
```

...ou, d'une manière plus générale :

```
<!ENTITY % xml-space "xml:space (default |preserve) 'preserve'">
```

```
<!ATTLIST unElement
    %xml-space;>
```

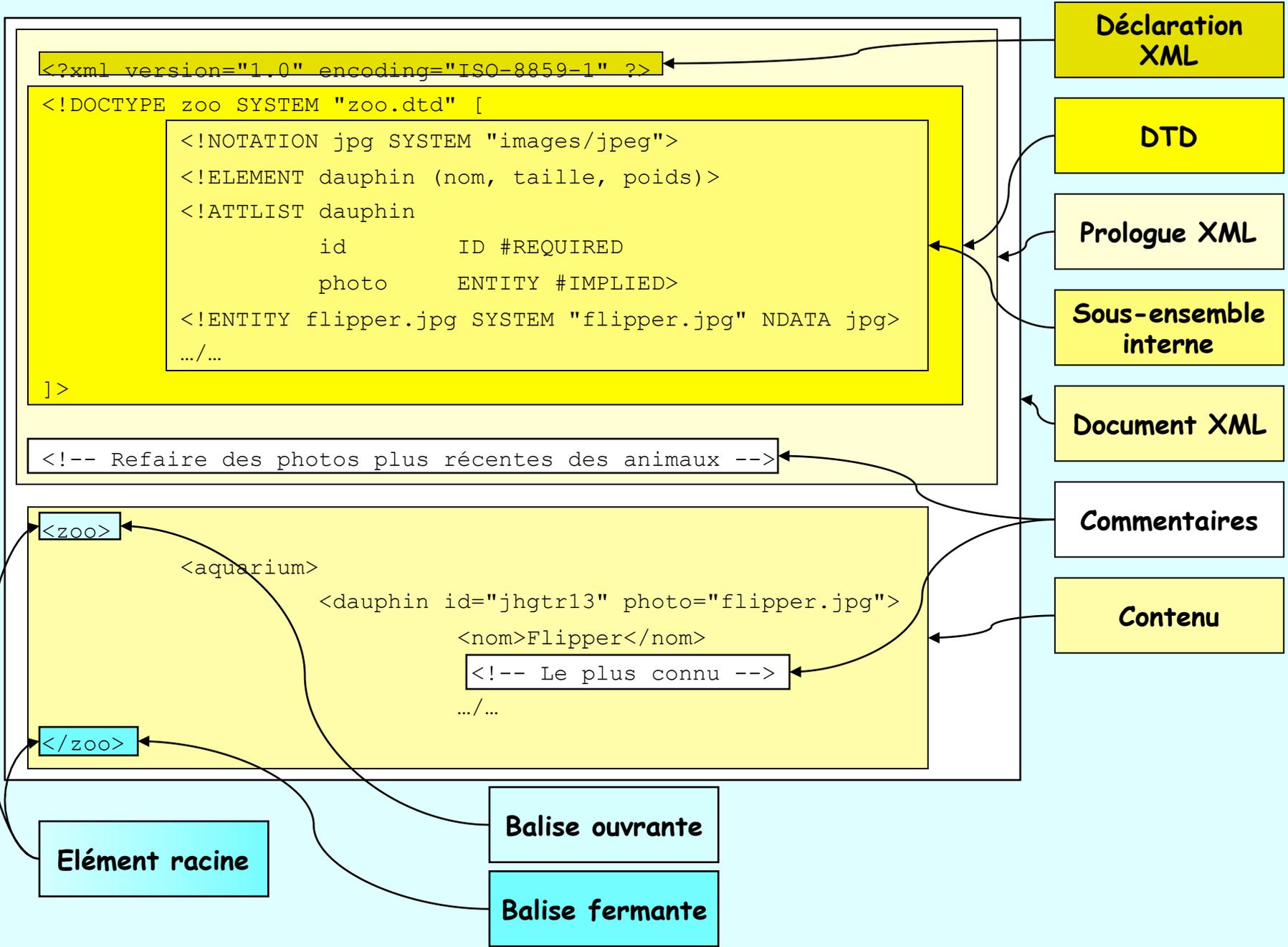
xml:space
est destiné
aux
applications
XML, pas aux
processeurs
XML

Vérification des contraintes de forme d'un document

- Un seul élément racine pour le document
- Toute balise ouverte doit être fermée
- Les balises doivent être correctement imbriquées
- Un élément ne doit pas avoir 2 attributs avec le même nom
- Les valeurs des attributs doivent être entre guillemets (simples ou doubles)
- Les commentaires et instructions de traitement ne doivent pas apparaître à l'intérieur des balises
- Les caractères < et & doivent être échappés dans les données textuelles d'un élément ou d'un attribut
- .../...

Le parseur est obligé de signaler les erreurs, il doit ensuite s'arrêter

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Cours SYSTEM "Cours.dtd">
<Cours>
  <Titre>Cours XML</Titre>
  <Auteur>
    <Nom>Poulard</Nom>
    <Prénom>Philippe</Prénom>
  </Auteur>
  <Description>
    Ce cours aborde les concepts de base mis en œuvre
    dans XML.
  </Description>
</Cours>
```

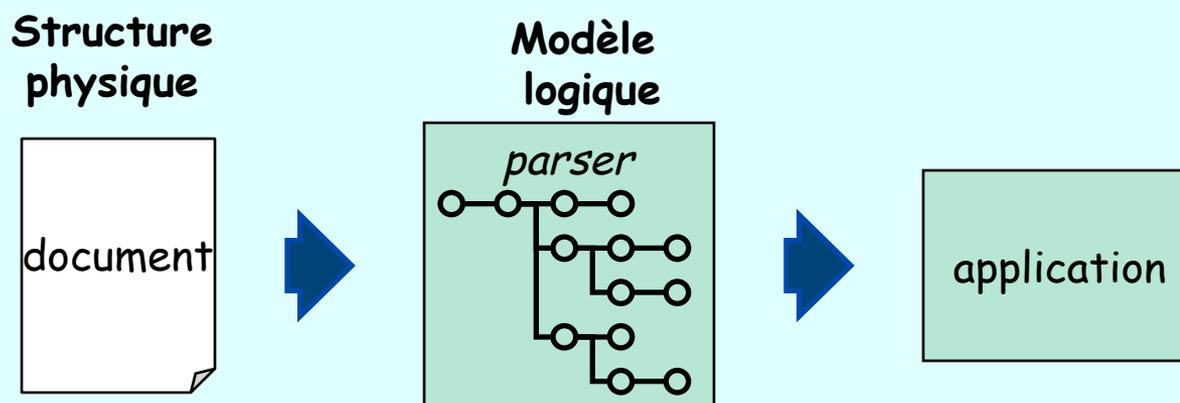


Un parseur (*parser*) est un programme informatique qui permet de vérifier :

- que la syntaxe de la DTD est bien conforme à la spécification XML,
- que l'instance du document est conforme à la DTD.

Parseur = analyseur syntaxique

Outil permettant de lire un document XML et de transmettre à une application le contenu et la structure **sous la forme d'un modèle logique**



Les parsers sont déterministes :

La **spécification XML interdit** au parseur de faire des suppositions sur la structure au cas où il rencontrerait des ambiguïtés

Dès qu'un document est erroné, il doit le signaler et s'arrêter

La spécification XML précise quelles doivent être les erreurs bloquantes

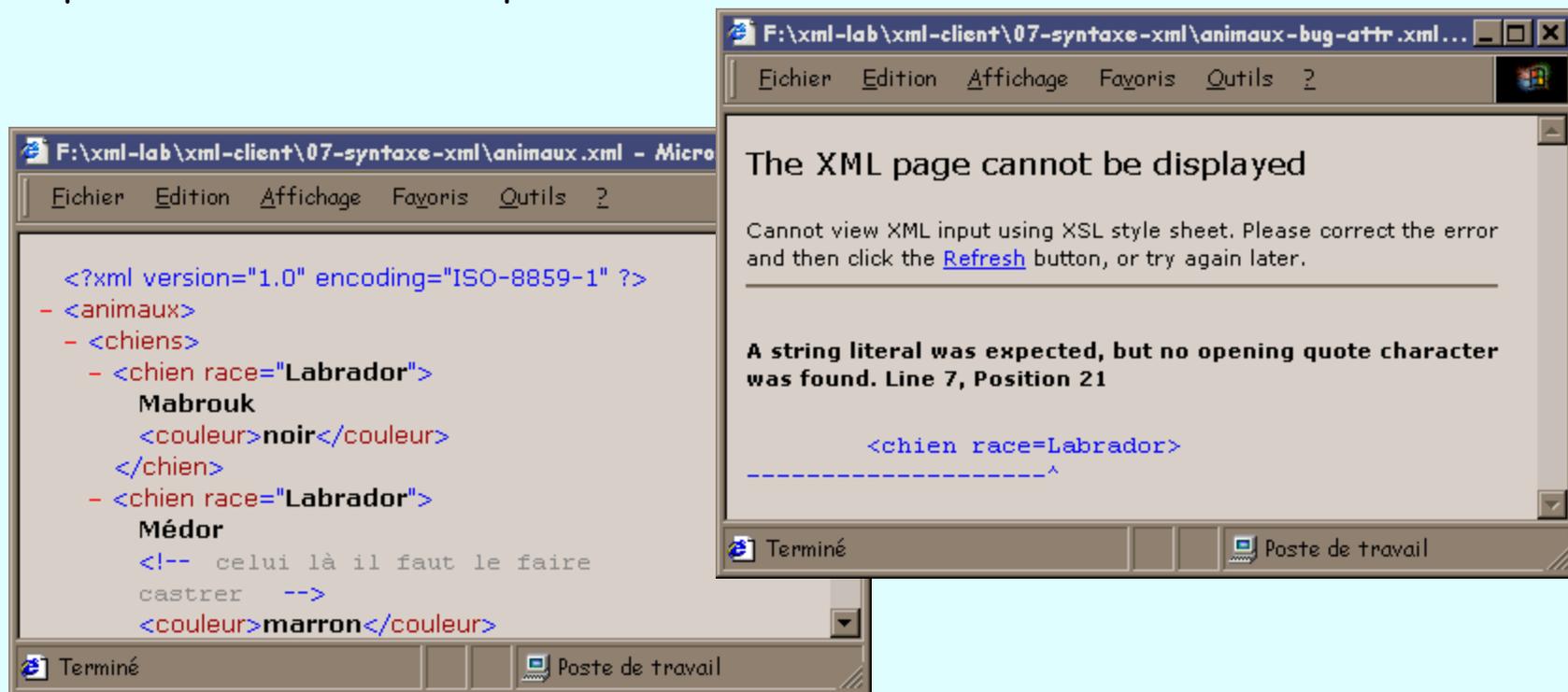
En passant par un parser, une application est sûre de ne traiter que des documents bien formés ou valides.

En le chargeant dans un navigateur Web

- parser intégré (mais attention aux prises de liberté)
- visualisation arborescente du document
- possibilité d'associer une feuille de style CSS ou XSL au document
- utile pour une vérification rapide d'un seul document



attention à la compatibilité

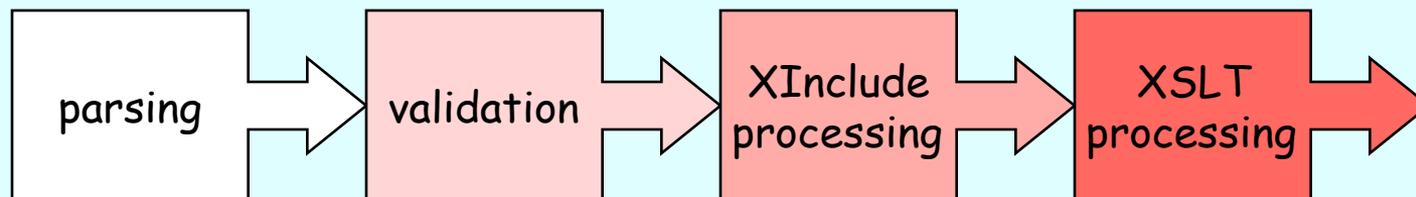


En utilisant un outil

- Ce sont souvent des bibliothèques exécutables
- Utilisable par une commande en ligne
- Utilisable par des applications
- Utile pour des traitements complets, comme la publication

Core XML standards : il manque la possibilité d'appliquer des chaînes de traitement à un document

Par exemple :



Les techniques existantes :

- La déclaration `DOCTYPE`
- Le pseudo-attribut `standalone`
- L'attribut `xsi:schemaLocation`
- L'instruction de traitement `xml-stylesheet`
- ...

Nouvelles technos XML :

- XSP (Cocoon)
- Active Tags : XCL

abordent le problème avec, chacune, une approche différente

Il faut une solution généraliste et extensible, qui puisse ne pas être limité au document seul : les traitements doivent être déterminés à la fois par des informations internes et externes au document qui doit être traité

Besoin d'une "pull API" qui permette des accès aléatoires

⇒ XML 2.0 (ce qui n'est pas prêt d'arriver)

Editeurs de texte

vi
emacs
notepad

Editeurs XML

→ pour les utilisateurs finaux
(masquage de la complexité de XML)

Adobe FrameMaker, www.adobe.com

XML Pro, www.vervet.com

XML Writer, xmlwriter.net

XML Notepad, msdn.microsoft.com/xml/notepad/intro.asp

Xmetal, SoftQuad, xmetal.com

XML Spy, www.xmlspy.com

Epic, ArborText, www.arbortext.com

XXE, XML Mind, www.xmlmind.com

Adept Editor - FT12291.sgm

Fichier Edition Rechercher Affichage Insérer Entités Objet Tableau Outils Options Format Fenêtre Aide

Rechanges :

2310 14 491 8573	Quantité : 1
------------------	--------------

Ingrédients :

8010 14 428 0821	Quantité : SB
8010 14 428 0822	Quantité : SB

Opération principales :

Montage de l'interface affût AA 53T2 sur le plateau du VLRA

- S'il y a lieu débâcher le véhicule, démonter les bancs et la ridelle arrière
- Déposer
 - la roue de secours (lot de bord)
 - la plaque minéralogique arrière gauche

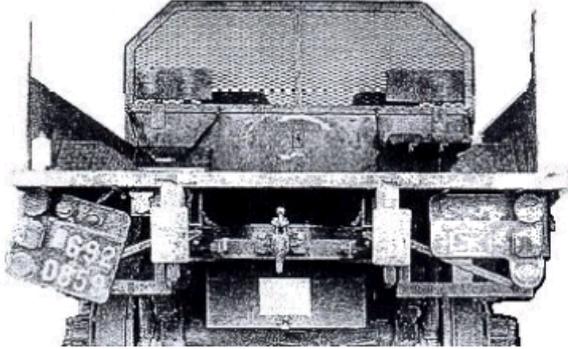


Illustration : FT.12291.1 013 A. Plaque minéralogique

dtmodg module proc proced preltreqs supplyli | [fmt nécessaire] EXT MOD INC

Epic
(Adept Editor)

Environnements
d'éditeurs à
personnaliser (au
profit d'une DTD)

XXE

document=content=chapter=section=p=#text

ARTICLEturing.1.0.xml : Interstices (collapsible sections and chapters) : Interstices

Date de création: 8/14/03
 Règle: NONE
 Date de mise à jour:
 validity-begin-date:
 validity-end-date:
 Liste des versions:
 # no: 1.0, Etat: PUBLISH, Date: 2003-08-14
 Description:
 ©INRIA 2003
 Abstract:
 Peut-on tout calculer ? Toute propriété mathématique est-elle décidable ? Ces questions ont passionné les mathématiciens bien avant les premiers ordinateurs. À l'âge de 24 ans, le britannique Alan Mathison Turing, mathématicien génial, entre autres, imagine un concept de machine théorique. Il établira une correspondance entre les notions de calculable et de programmable sur cette machine imaginaire, ainsi qu'avec celle de décidabilité.

▶ Section 1:

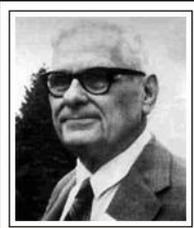
▼ Chapter 1: De Church à Turing

▼ Section 1: Calcul et calculabilité

Ainsi, toute multiplication de nombres entiers peut être réduite à des additions successives. De même toute division se ramène à des additions et des soustractions. Quant aux additions et soustractions, elles se convertissent aisément en manipulations de symboles ou de cailloux. En revanche, nous avons l'impression que d'autres fonctions définies formellement semblent difficiles, voire impossibles à « caillouter ». Autrement dit, à décrire uniquement en termes de déplacements de cailloux.

Jusqu'où de telles manipulations peuvent-elles aller ? Quel est l'ensemble des fonctions f , exprimables en termes mathématiques, pour lesquelles il existe une succession de manipulations de symboles déterminant sans ambiguïté l'image de n importe quelle valeur x par f ? C'est-à-dire un calcul de la valeur y telle que $y = f(x)$? Autrement dit, quelle est la classe des fonctions calculables ? Telle est la question qu'aborde la calculabilité.

▼ Section 2: La thèse de Church

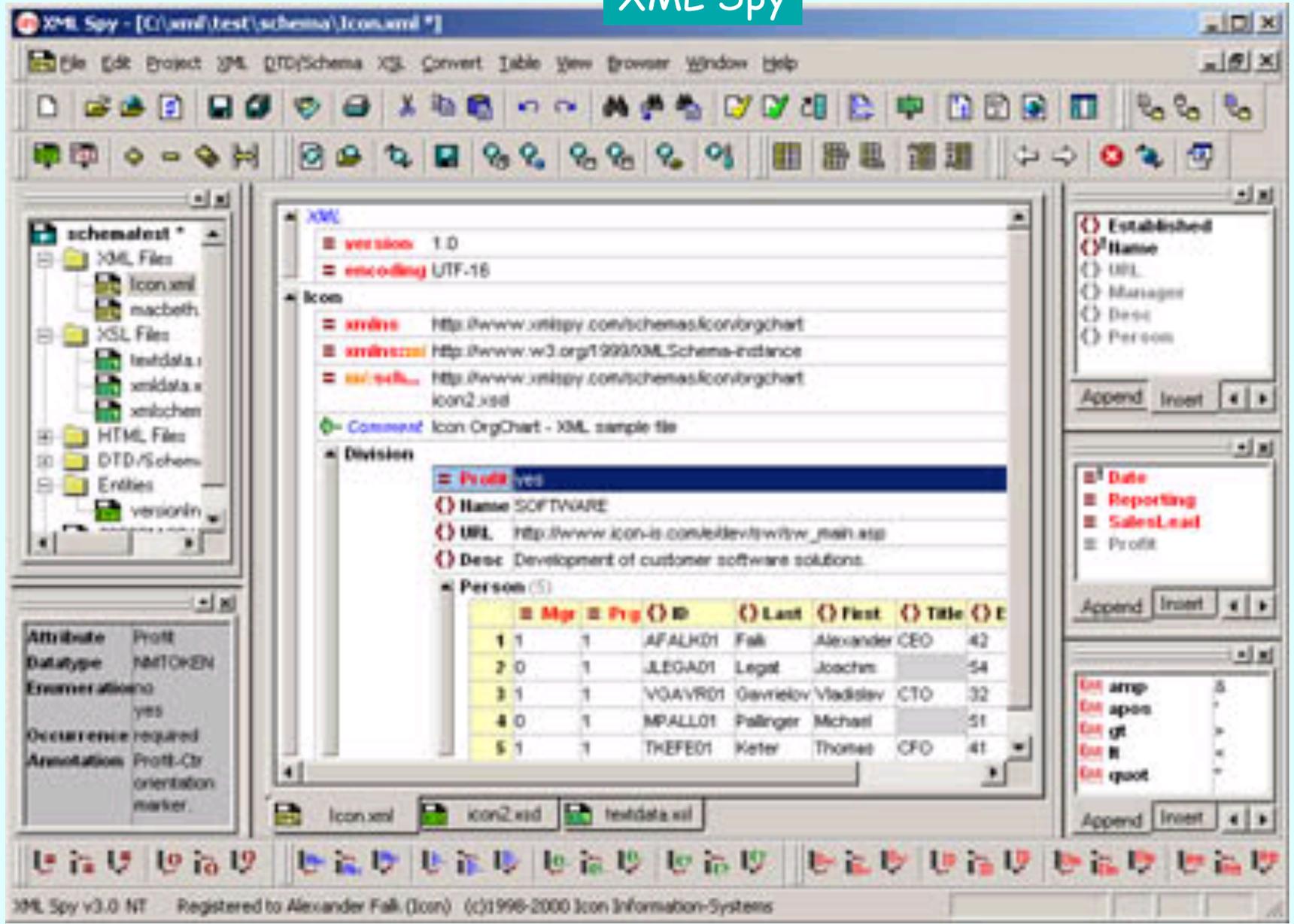


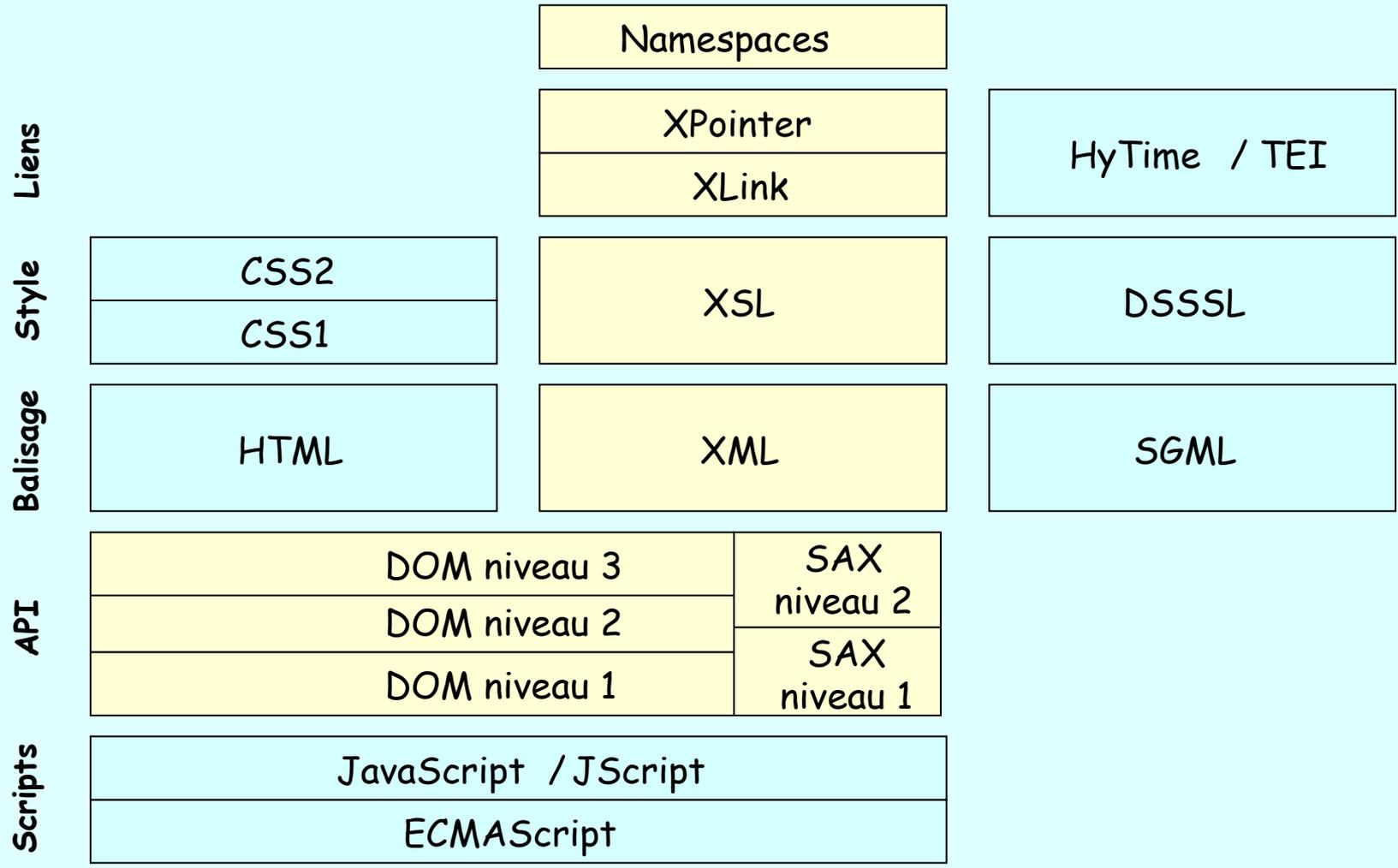
Attributes: Validity Spell
 Search Characters Edit

#text

XML Spy

Fondamentaux XML





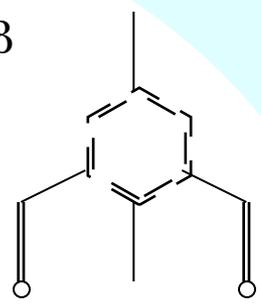
XSLT, XSLFO, WXS, XLink...

Applications liées au standard
 Applications « métier »
 Applications propriétaires

MathML

$$\partial_{\alpha} F^{\alpha\beta} = \frac{4\pi}{c} J^{\beta}$$

CML



SMIL



WML

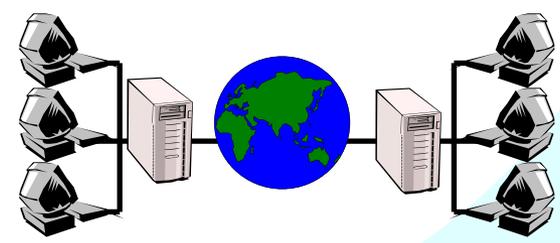


SOAP

XML-EDI

XHTML

XForms



...

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

Structures de données

utiliser XML car il y a les outils pour :

- adresser les données avec XPath
- les manipuler avec le DOM
- transformer les structures avec XSLT

et intrinsèquement :

- échange
- stockage

Fichiers de configuration

- facilité d'extension
- portabilité
- choix du langage qui utilise un parseur

→ inutile d'analyser soit-même les données
→ utilisation d'un parseur

Edition structurée

→ approche fonctionnelle du document

- préoccupations du contenu, pas de la forme
- publication cross-media
- traitement des données

documentation technique :
• XML est présent dans de nombreux secteurs de l'industrie

Utilisations de XML

- structures de données en arbre
- information semi-structurée
- structures documentaires

Élément vs Attribut

Quelle est la meilleure structure ?

```

<dauphin>
  <nom>Flipper</nom>
  <sexe>M</sexe>
  <taille unité="cm">215</taille>
  <poids unité="kg">105</poids>
  <date-naissance>
    <année>1997</année>
    <mois>4</mois>
    <jour>1</jour>
  </date-naissance>
</dauphin>

```

```

<dauphin nom="Flipper"
  sexe="M"
  taille="215"
  unité-taille="cm"
  poids="105"
  unité-poids="kg"
  année-naissance="1997"
  mois-naissance="4"
  jour-naissance="1"
/>

```

```

<animal espèce="dauphin">
  <nom>Flipper</nom>
  <sexe>M</sexe>
  <taille unité="cm">215</taille>
  <poids unité="kg">105</poids>
  <date-naissance>
    <année>1997</année>
    <mois>4</mois>
    <jour>1</jour>
  </date-naissance>
</animal>

```

```

<animal espèce="dauphin"
  nom="Flipper"
  sexe="M"
  taille="215"
  unité-taille="cm"
  poids="105"
  unité-poids="kg"
  année-naissance="1997"
  mois-naissance="4"
  jour-naissance="1"
/>

```

```

<date-naissance
  année="1997"
  mois="4"
  jour="1" />

```

Les données sont ordonnées → **ne peut être un attribut**

Les données contiennent des sous-structures → **ne peut être un attribut**

Les données sont sur plusieurs lignes → devrait être un élément

Les données doivent être mises à jour → devrait être un élément

Les données sont parmi un petit nombre de possibilités → devrait être un attribut

Les données sont de petites chaînes qui changent rarement → devrait être un attribut

Visibilité

Si les données doivent être publiées, il vaut mieux les mettre dans un élément.

Si les données sont des paramètres ou des méta-données, il vaut mieux les mettre dans un attribut.

Exemple avec SVG

Atomicité des données :

```
<polygon fill="blue" stroke="green" stroke-width="12">
  <point x="350" y="75"/>
  <point x="379" y="161"/>
  <point x="469" y="161"/>
  <point x="397" y="215"/>
  <point x="423" y="301"/>
  <point x="350" y="250"/>
  <point x="277" y="301"/>
  <point x="303" y="215"/>
  <point x="231" y="161"/>
  <point x="321" y="161"/>
</polygon>
```

Oui

Adopté par le W3C ☹️

```
<polygon style="fill:blue; stroke:green; stroke-width:12"
  points="350,75 379,161 469,161 397,215 423,301
  350,250 277,301 303,215 231,161 321,161">
</polygon>
```

Non

En fait,
OUI aussi
dans ce cas
mais à
éviter

Propriété composée (*shorthand property*) ➡ à éviter

Oblige à parser sans parseur standard
(non XML parseur dans un parseur XML)

Atomicité des données :

- données élémentaires = parser XML
- données non atomiques = parser non XML ou données non parsable

Transformation XSLT :

- sans perte sémantique
- avec perte sémantique
- avec ajout sémantique
 - nécessite de parser et interpréter
 - pas toujours réalisable (ou souvent irréalisable)

Structures connues et parsables

Numéro de Secu

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

→ Parser de no de Secu

```
<insee>
  <sexe>1</sexe>
  <année>69</année>
  <mois>06</mois>
  <dept>13</dept>
  <nville>001</nville>
  <nnaiss>084</nnaiss>
</insee>
```

Numéro de nomenclature OTAN

```
<nno>1234 14 1234567</nno>
```

→ Parser de NNO

```
<nno>
  <groupe>12</groupe>
  <classe>34</classe>
  <nation>14</nation>
  <no>1234567</no>
</nno>
```

Structures non parsables

```
<auteur>Philippe Poulard</auteur>
<auteur>Jean De la Fontaine</auteur>
<auteur>Jean Paul Marcel</auteur>
```



```
<auteur>
  <prénom>Philippe</prénom>
  <nom>Poulard</nom>
</auteur>
<auteur>
  <prénom>Jean</prénom>
  <nom>De la Fontaine</nom>
</auteur>
<auteur>
  <prénom>Jean</prénom>
  <nom>Paul Marcel</nom>
</auteur>
```

À éviter

Attention au sens : {
 3 février 2001 (au Japon)
 1^{er} février 2003 (en Europe)
 2 janvier 2003 (aux USA)

```
<date>01/02/03</date>
```

Nécessité de parser (parser de date)

Affichage souhaité :

1^{er} février 2002

=> parser de date

Signification
convenue pour
01/02/03

```
<date format="JJ/MM/AAAA">01/02/2002</date>
```

Indication de format au parser de date

```
<date>
  <année>2002</année>
  <mois>2</mois>
  <jour>1</jour>
</date>
```

- Forme atomique des données
- + Pas besoin de parser spécifique
- + Possibilité d'adressage par Xpath
- + Pas d'ambiguïté
- Forme verbeuse

Date au format ISO 8601
(C'est très formel !!!)

```
<!ATTLIST date format NOTATION (time) "time">
```

```
<!NOTATION time PUBLIC "ISO 8601:2000//NOTATION
Data elements and interchange formats -
Information interchange-
Representation of dates and time//EN" >
```

Inconvénient : le format ISO 8601 permet aussi de coder les périodes et permet des imprécisions (pour exprimer par exemple "le mois de juin de l'année 2002")

Comment limiter son utilisation dans un contexte donné ?

↳ Lorsqu'on ne veut qu'une simple date jj/mm/aa ?

Une autre solution : être conforme avec XML Schema Part 2 : Datatypes

Autres préoccupations :

Quel est le meilleur format pour les balises composées de multiples mots ?

- date-de-naissance
- dateDeNaissance
- DateDeNaissance
- date_de_naissance

Élément vs attributs

Quand créer un espace de nommage ?

- Pour chaque nouveau projet ?
- Pour chaque nouvelle classe de document ?

Faut-il que les attributs soient globaux ?

(qu'ils aient le même sens dans des éléments différents ?)

Introduire des éléments factorisants → container pratique

Foos

Foo
Foo

Versions

Version
Version
Version

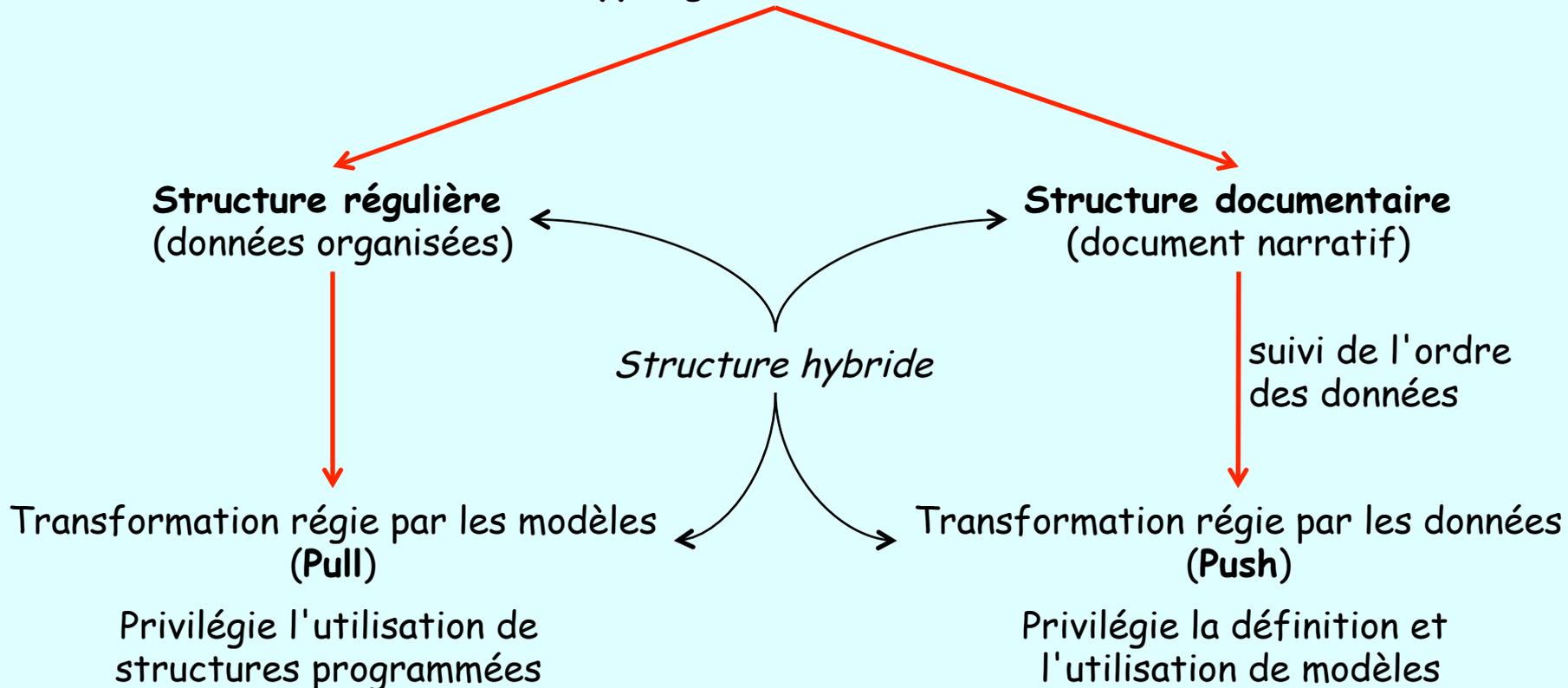
Voitures

Voiture
Voiture

```
<commande id="kh34jl">
  <article id="4l34m" prix="120" qté="5">
  <article id="df40t" prix="210" qté="1">
  <article id="lnk23" prix="330" qté="4">
</commande>
```

```
<vers>Maître <b>corbeau</b> sur un
arbre perché tenait dans son bec un
fromage</vers>
<vers>Maître <i>renard</i> par l'odeur
alléché .../...</vers>
```

Typologie de structure



La plupart des documents sont des structures hybrides. Les structures purement documentaires ou régulières se trouvent dans des cas particuliers