Bases de données documentaires et distribuées
Cours NFE04
Codage XML

Auteurs : Raphaël Fournier-S'niehotta, Philippe Rigaux, Nicolas Travers
prénom.nom@cnam.fr

Département d'informatique
Conservatoire National des Arts & Métiers, Paris, France

XML, format pour données "semi-structurées"

XML est d'abord un standard du World-Wide-Web Consortium (W3C).

- La sérialisation de documents XML est normalisée, ce qui permet des échanges réseau sans perte d'information.
- XML est un format générique, qui se spécialise ensuite en **"dialectes"** pour des domaines spécifiques (e.g., XHTML, affichage dans un navigateur).
- Le W3C a défini des standards associés : DOM (le "modèle"), XSchema (le typage), XPath (navigation), XSLT (restructuration), XQuery (requêtes), et beaucoup d'autres. Très compliqué, et surtout très lourd.

Remarque

XML est une version simplifiée de **SGML**.

Documents XML : structure et contenu

XML permet de **structurer** du contenu textuel.

Le contenu textuel :

> Le film de SF Gravity, réalisé par Alfonso Cuaròn, est paru en l'an 2013.

Le même contenu, structuré en XML.

```
<?xml version="1.0" encoding="utf-8"?>

<film>
  <titre>Gravity</titre>
  <annee>2013</annee>
  <genre>SF</genre>
  <realisateur
     nom="Cuaron"
     prenom="Alfonso"/>
</film>
```

Documents XML : modèle et sérialisation

Modèle Un document XML est un arbre, chaque nœud a un type particulier.

Sérialisation tout document XML peut être transformé en une chaîne de caractères représentant l'arborescence du document.

La norme XML définit une **syntaxe** pour coder des documents textuels hiérarchiques. Aucune interprétation n'est définie à priori.

Les dialectes XML imposent des contraintes, et définissent une interprétation.

Le dialecte XHTML

Un document XHTML est un document XML. Il obéit à des contraintes de structure, et s'interprète sous forme de règles d'affichage à l'écran.

Qui utilise XML ?

XML est le standard (W3C) pour la représentation de données textuelles.

Exemples : XHTML, DocBook, RSS, ...

XML est AUSSI utilisé pour sérialiser des données applicatives, par exemple pour les données gérées par vos applications personnelles.

Exemples : Traitements de texte (Word), tableurs (Excel), calendriers, graphiques (en SVG), etc.

Dans tous les cas

La représentation en XML permet de **réutiliser** le contenu, de **l'échanger** avec d'autres applications.

Exemple : flux de données RSS

Pour disséminer des informations en mode "push" (cf. Blogs, sites de news, etc.).

```
<?xml version="1.0" encoding="UTF-8" ?>

<rss version="2.0">
<channel>
<title>Webdam Project</title>
<atom:link href="http://webdam.inria.fr/wordpress/?feed=rss2"
              rel="self" type="application/rss+xml" />
<link>http://webdam.inria.fr/wordpress</link>
<description>Web Data Management</description>
<pubDate>Wed, 26 May 2010 09:30:54 +0000</pubDate>

  <item>
    <title>News for the beginning of the year</title>
    <description>...</description>
    <link>http://webdam.inria.fr/wordpress/?p=475</link>
    <pubDate>Fri, 15 Jan 2010 08:48:45 +0000</pubDate>
    <dc:creator>Serge</dc:creator>
    <category>News</category>
  </item>
</channel>
</rss>
```

Scalable Vector Graphics (SVG)

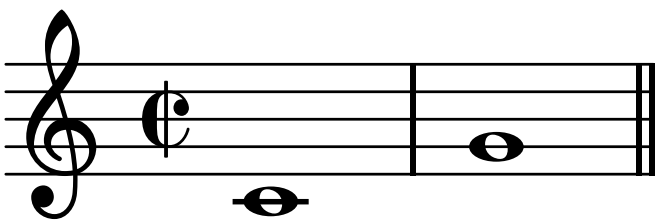
Et oui, on peut représenter des données graphiques en XML !

```
<?xml version="1.0" encoding="UTF-8" ?>
<svg xmlns="http://www.w3.org/2000/svg">

    <polygon points="0,0 50,0 25,50"
          style="stroke:#660000;"/>

  <text x="20" y="40">Some SVG text</text>
</svg>
```

Music XML

Ou des partitions musicales.

```
<?xml version="1.0" encoding="UTF-8"?>

<score-partwise version="2.0">
  <part id="P1">
      <attributes>
        <divisions>1</divisions>
      </attributes>
      <note>
        <pitch>
          <step>C</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
      </note>
  </part>
</score-partwise>
```



Syntaxe XML

Un document XML est composé de :

Un prologue

1. déclaration de document XML
   `<?xml version = "1.x" [encoding = "norme" ]? >`
2. suivie éventuellement (sans contrainte d'ordre)
3. d'une référence à une DTD
   `<!DOCTYPE element racine SYSTEM "uri" >`
4. de commentaires `<!-- texte -->`
5. d'instructions de traitement (par ex. appel à XSLT)
   `<?instruction ... ? >`

Un corps

- une imbrication d'**éléments** et de **textes**

Quelques exemples de base

Remarque : les retours à la ligne n'ont aucune signification.

```
<document/>

<document> Hello World! </document>

<document>
  <salutation> Hello World! </salutation>
</document>

<?xml version="1.0" encoding="utf-8" ?>
<document>
  <salutation color="blue"> Hello World! </salutation>
</document>
```

Le dernier exemple est le seul complètement correct : il comprend le prologue.

Forme hiérarchique des documents : le modèle DOM

Un document XML est interprété comme un **arbre**, basé sur un modèle, le DOM (**Document Object Model**) normalisé par le W3C.

Dans le DOM, chaque nœud de l'arbre a un **type**, et une **description** (nom, contenu)

Les nœuds de type **Element** correspondent au balisage dans la forme sérialisée.

1. ils définissent la **structure** du document.
2. ils ont un **nom** (mais pas de **valeur**)
3. ils ont un **contenu** : le sous-arbre dont ils sont racine.

Les nœuds de type **Text** correspondent au **contenu textuel** du document.

1. ils constituent les feuilles de l'arborescence (un nœud **Text** n'a pas de fils) ;
2. ils ont une **valeur** (mais pas de **nom**).

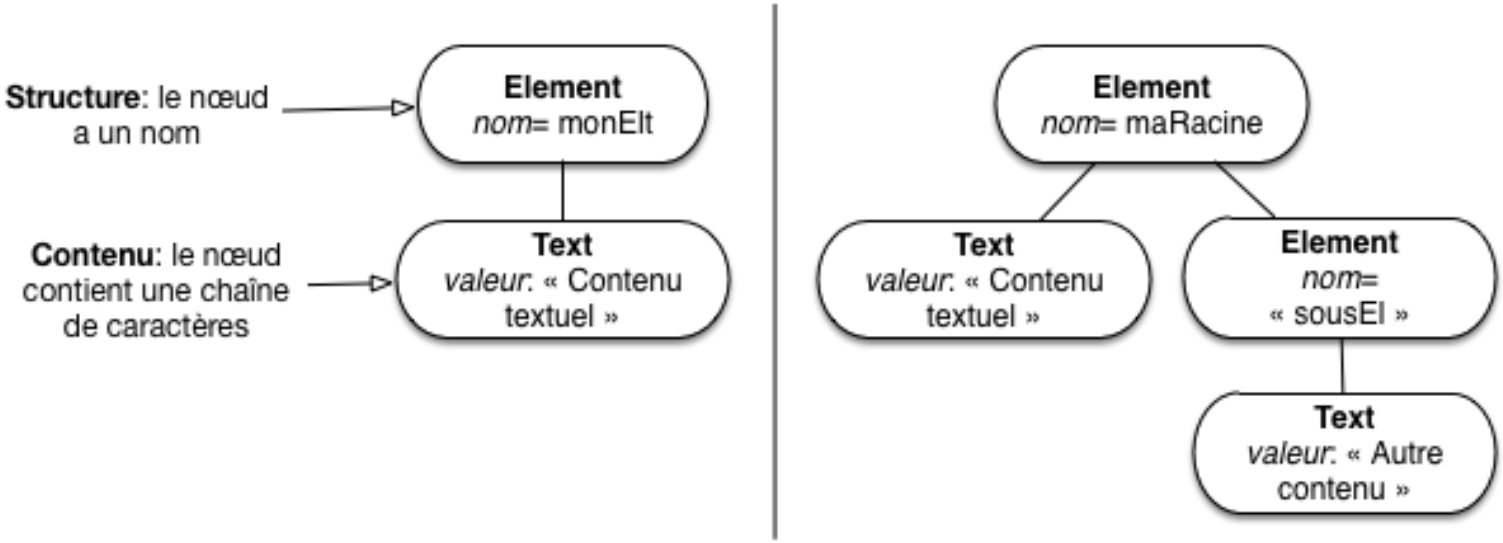
Correspondance forme sérialisée - forme DOM

Voici deux exemples de base illustrant la construction élément/texte.

```
<monElt>
  Contenu textuel.
</monElt>
```

```
<maRacine>
  Contenu textuel
  <sousEl>
    Autre contenu
  </sousEl>
</maRacine>
```

Et la représentation arborescente (DOM).



Quelques remarques en vrac (anecdotique)

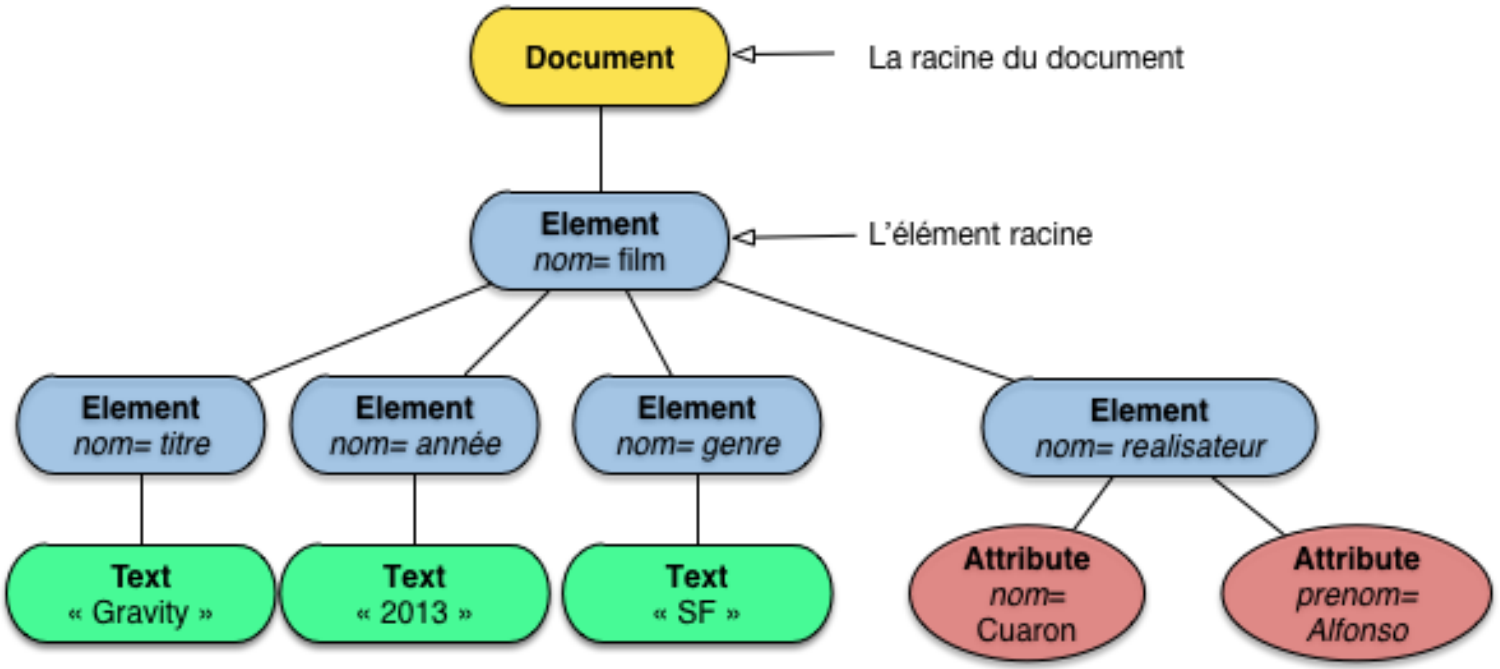
- Les sous-éléments sont ordonnés.
- Pas de caractères spéciaux (mais -,_,.,/ autorisés) dans les noms des éléments, pas d'espaces.
- Un élément peut être vide : `<nom_elt></nom_elt>`
  Notation abrégée : `<nom_elt/>` (Exemples d'éléments vides (XHTML) : `<br/>` ou
  `<img width="100" height="57" src="logoCnam.jpg"/>`)
- (Contrairement à HTML,) XML est *case sensitive* donc `<genie>` ≠ `<Genie>`
- Les caractères spéciaux peuvent être remplacés par une **référence** : `&` doit être remplacé par sa référence `&amp;` (version numérique : `&#28;`)
  < devient `&lt;` (ou `&#60;`)
  > devient `&gt;` (ou `&#62;`)
  " devient `&quot;` (ou `&#39;`)
  ' devient `&apos;` (ou `&#34;`)

D'autres types de nœud : **Attribute** et **Document**

Les **attributs** sont des paires clé/valeur attachées à un élément.
Un attribut a une **valeur** : un chaîne de caractères.

Un document XML sous forme sérialisée commence **toujours** par un prologue représenté par un nœud de type **Document**.

```
<?xml version="1.0"encoding="utf-8"?>
```



Résumé de l'essentiel

Forme sérialisée

- Un document débute par un prologue
- Le contenu est enclos dans une unique balise ouvrante/fermante.
- Chaque balise ouvrante `<nom>` **doit** avoir une balise fermante `</nom>` ; tout ce qui est entre les deux est soit du texte, soit du balisage correctement ouvert/fermé.

Forme arborescente (DOM)

- Un document est un **arbre** avec une racine (du document) de type **Document**
- La racine du document a un seul **élément** fils, de type **Element**, appelé **l'élément racine**.
- Chaque **Element** est un sous-arbre représentant du contenu structuré formé de nœuds **Element**, de nœuds **Text**, et parfois de nœuds **Attribute**.

Document bien formé

Un document est bien formé s'il respecte la syntaxe XML.

- à toute balise ouvrante est associée une balise fermante,
- les éléments sont imbriqués (pas de superposition),
- présence d'**un seul élément racine**,
- etc (se référer à la syntaxe XML).

Un document XML se doit d'être bien formé (sinon inexploitable).

Pour tester si l'un de vos documents est bien formé : un vérificateur syntaxique est disponible à l'adresse `http://www.w3schools.com/xml/xml_validator.asp`.
Vous pouvez également ouvrir votre fichier XML avec un navigateur web, intégrant généralement un parser XML (pour pouvoir être parsé, un document doit être bien formé).

Les entités et les références

Les entités sont des symboles qui désignent des fragments. Utile pour réutiliser du contenu, sans le répéter.

Une entité est **déclarée** (dans le prologue), puis **référencée**.

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

<!DOCTYPE a [
  <!ENTITY monNom "Charles Martel">
  <!ENTITY maSignature SYSTEM "signature.xml">
]>

<a>
  Mon nom est &monNom;.

  &maSignature;
</a>
```

Entités prédéfinies

Tout un ensemble de symboles qui sont interprétées comme du marquage / balisage si on les utilise directement.

Si on les veut **littéralement**, il faut utiliser des références d'entités.

Déclaration	Référence	Symbole.
`<!ENTITY lt "&#60;">`	`&lt;`	<
`<!ENTITY gt "&#62;">`	`&gt;`	>
`<!ENTITY amp "&#38;">`	`&amp;`	&
`<!ENTITY apos "&#39;">`	`&apos;`	'
`<!ENTITY quot "&#34;">`	`&quot;`	"

Commentaires et instructions de traitement

Les **commentaires** peuvent être placés n'importe où dans la forme sérialisée.

```
<!--C'est un commentaire -->
```

Ils apparaissent dans l'arbre DOM comme des nœuds de type **Comment**.

Remarque

Très peu d'intérêt : on n'est pas censé *lire* un document XML. Peut être remplacé par un élément de type **element** et de nom `comment`.

Instructions de traitements : commandes spécifiques à une application particulière.

Typiquement, on demande à un processeur XSLT d'utiliser un programe `prog.xslt` :

```
<?xml-stylesheet href="prog.xslt"type="text/xslt"?>
```

Sections litérales

Un analyseur (parseur) XML cherche à analyser tout le contenu pour détecter du marquage structurel.

Problème : et si on ne veut **pas** que le contenu soit analysé ?

```
<?xml version='1.0'?>
<program>
if ((i < 5) && (j > 6))
   printf("error");
</program>
```

Solution : soit on utilise des entités (lourd), soit on insère la texte à protéger dans une **section littérale**.

```
<?xml version='1.0'?>
<program>
<![CDATA[if ((i < 5) && (j > 6))
           printf("error");
]]>
</program>
```

C'est tout pour l'instant !

Ce qui précède suffit pour produire et comprendre à peu près tous les documents XML.

Autres aspects qui peuvent (quand même) s'avérer importants :

1. **Validation d'un document**. Comme déclarer qu'un document doit être conforme à une structure donnée ? Comment le vérifier ?
   ⇒ typage de documents XML, avec les DTD ou XML Schéma (horreur !)
2. **Conflits de nom**. Comment bien interpréter une balise (nom d'élément) quand on assemble plusieurs documents ?
   ⇒ les *namespaces* (espaces de noms).
3. et autres aspects exotiques...

Ce qu'il faut retenir

XML = tentative la plus complète de formaliser un modèle de représentation semi-structuré.

- Très complet.
- Très lourd.... Syntaxe compliquée et redondante, éléments/attributs, beaucoup de détails peu utiles.

L'essentiel :

- La structuration en arbre ; l'existence d'une syntaxe pour coder cet arbre (la forme sérialisée) ;
- La notion de **dialecte**, ou **espace de nom** : des contraintes de nommage et de structuration pour lesquelles les documents ont une interprétation.
- JSON peut être vu comme une forme ultra-simplifiée ; tend à remplacer XML (sauf pour les documents dits "rédactionnels").