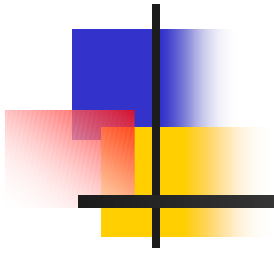


NSY107 - Intégration des systèmes client-serveur

Cours du 10/06/2006, 4 heures,

Thème : XML

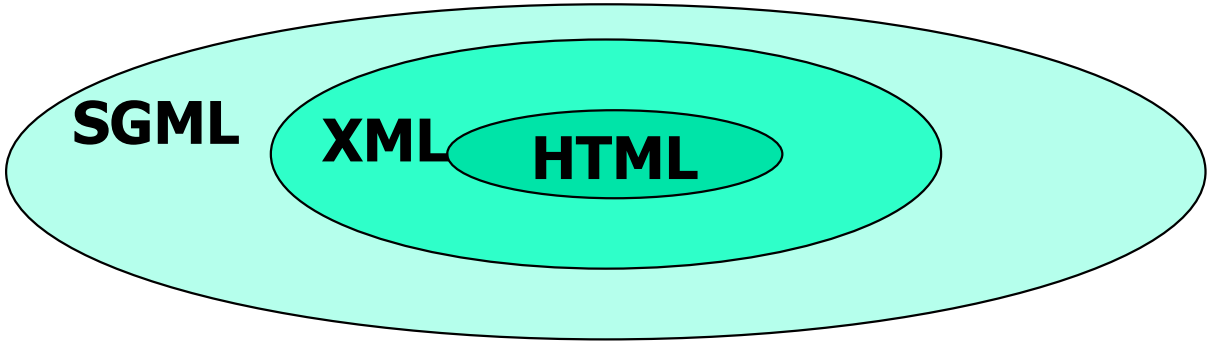
© Emmanuel DESVIGNE
<emmanuel@desvigne.org>

Plan du cours « XML »

- Introduction/Généralités ;
- Codage des caractères internationaux ;
- De l'intérêt du XML ;
- DTD ;
- TP1 ;
- Initiation aux schémas XML (XSD) ;
- TP2 ;

Introduction/Généralités [1/10]

- XML = « Extensible Markup Language » ➔ langage de balises – TAG – (comme le HTML), mais extensible (dont on peut définir la grammaire)
- XML est issu de SGML (Standard Generalized Markup Language), un métalangage standard international



Introduction/Généralités [2/10]

- XML est défini par le W3C (World Wide Web Consortium), consortium fondé en 1994 pour promouvoir la compatibilité des technologies du web. Le W3C n'émet pas des normes, mais des recommandations.
- La gestion du W3C est assurée conjointement par le Massachusetts Institute of Technology (MIT) aux EUA, le European Research Consortium for Informatics and Mathematics (ERCIM) en Europe, et l'Université Keio au Japon.
- Site web : http://www.w3c.org

Introduction/Généralités [3/10]

- Exemple :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Livres><Livre>
    <Titre>Histoire de la pensée</Titre>
    <Auteur>L. Jerphagnon</Auteur>
    <Auteur>J. L. Dumas</Auteur>
    <Edition Editeur="TALLANDIER"
ISBN="2235018548" Annee_1ere_par="1990" />
    <Biographie Auteur="L. Jerphagnon">Pr des
Universités, a été 18 ans conseiller auprès de
l&apos;Institut international de philosophie. Il
est l&apos;un des membres fondateurs du Centre
international d&apos;études platoniciennes et
aristotéliciennes d&apos;Athènes, et membre de
l&apos;Académie nationale
d&apos;Athènes.</Biographie>
</Livre></Livres>
```

Introduction/Généralités [4/10]

- Quelques règles définissant le XML :
  - XML est « Case sensitive » (**`Titre` ≠ `titre` ≠ `TITRE`**)
  - Une balise est aussi appelée « *élément* »
  - Un flux XML doit toujours contenir un élément racine (« *Livre* » dans l'exemple précédent)
  - Toute balise doit être fermée : « **`<A>…</A>`** », ou encore « **`<A />`** » (la notation « **`<A />`** » équivaut à « **`<A></A>`** », i.e. une balise ouverte et immédiatement fermée), et peut contenir des attribus (ex : **`<Poids Mesure="4.2" Unite="Kg" />`**)
  - La valeur des attribus doit toujours être entre cote « **`blabla='…'`** » ou double cote « **`blabla= "…"`** »

Introduction/Généralités [5/10]

- Les balises doivent être correctement imbriquées (« `<P>bla <B> bla.</B></P>` » est correct, alors que « `<P>bla <B> bla.</P></B>` » ne l'est pas)
- Les noms d'éléments peuvent comporter des lettres, des chiffres, des tirets, des *underscores* « _ », des deux-points, ou des points. Le caractère deux-points « : » ne peut être utilisé que dans le cas particulier. De plus :
  - le nom doit toujours contenir au moins une lettre,
  - Le nom ne peut commencer par « XML » (quelle que soit la casse).

Introduction/Généralités [6/10]

- Les caractères « < », « > », « & », « ' », et « " » ne peuvent pas être utilisés dans le texte (car utilisés dans le balisage) ➔ on note :
  - « **<** » à la place de « < »,
  - « **>** » à la place de « > »,
  - « **&** » à la place de « & »,
  - « **"** » à la place de « " »,
  - et « **'** » à la place de « ' ».
- Les commentaires sont entre « **<!--** » et « **-->** ». Ils ne peuvent contenir des « **--** »

Introduction/Généralités [7/10]

- Les instructions de traitement (PI – Processing Instructions –) permettent aux documents XML de contenir des instructions destinées aux applications. Ex : « **<?perl code perl… ?>** », ou « **<?php code en php… ?>** »
- Les sections « CDATA » permettent de mettre du texte sans que les caractères spéciaux ne soient interprétés comme du balisage. Elles commencent par « **<![CDATA[** » et se terminent par « **]]>** ». La chaîne « **]]>** » ne doit pas figurer à l'intérieur d'un CDATA. Ex : « **<A><![CDATA[Le <B> ne sera pas vu comme une balise]]></A>** »

Introduction/Généralités [8/10]

- Un flux XML doit commencer par la version des spécifications XML employée. Ex : « **`<?xml version="1.0" ?>`**… »
- Lors de cette précision de N° de version, il est possible de préciser le système d'encodage des caractères (par défaut : UTF-8). Ex : « **`<?xml version="1.0" encoding="ISO-8859-15" ?>`**… »
- On dit qu'un flux XML est « bien formé » quand il respecte scrupuleusement toutes ces règles
- **Remarque :** le « **XHTML** » n'est finalement que de HTML « bien formé » (ce qui devrait toujours être le cas).

Introduction/Généralités [9/10]

- Un document XML est dit « *valide* » quand il est :
  - Bien formé,
  - Et qu'il respecte une grammaire (grammaire qui définit les noms des TAGs attendus, leur ordre, les attributs attendus pour chaque TAG, etc.).
- Deux principaux langages permettent de définir une grammaire d'un document valide :
  - DTD – Document Type Definition – (plutôt ancien),
  - le W3C XML Schéma (plus commun).

Introduction/Généralités [10/10]

- Un logiciel qui prend en paramètre une telle description de document valide (en DTD ou en XML schéma), qui analyse un document XML, et qui indique s'il est valide ou pas s'appelle un « parseur »/« parser » (analyseur syntaxique)
- Conclusion = XML peut être vu comme :
  - format de document,
  - format de données,
  - mode de structuration de l'information,
  - méta-langage (langage de description de langage).

Codage des caractères internationaux [1/5]

- Pour coder les caractères, historiquement, on a utilisé des tables associant un code sur 1 octet à un caractère. Exemples :
  - Table EBCDIC (IBM),
  - Table ASCII (utilise initialement 7 bits).
  - Les tables ASCII sur 8 bits utilisent la table des 128 caractères standards (table ASCII 7 bits), plus 128 caractères « étendus » ; on parle alors de « pages de code ». Ces dernières dépendent du pays :
    - page de code 437 : américain, graphique ;
    - page de code 850 : multilingue européen ;
    - ISO-8859-15 : Europe de l'ouest, avec le symbole €
    - etc.

Codage des caractères internationaux [2/5]

- Or, est apparu une multitude de pages de code, certaines étant définies par les éditeur d'OS ou de logiciels, d'autres étant normalisées, etc.
- Pour harmoniser tout ça : Unicode (qui a fini par se synchroniser avec le travail de normalisation ISO/CEI 10646)
- Unicode = jeu unique de caractères

Codage des caractères internationaux [3/5]

- L'actuelle version d'Unicode (vers. 4.1.0 de nov. 2005) contient près de 245'000 « points de codes » assignés dans un espace pouvant en contenir 1 114 112 différents (21 bits) :
  - 137'468 caractères à usage privé (assignés dans toutes les version d'Unicode et suffisants pour tous les usages) ;
  - plus de 97'755 lettres ou syllabes, chiffres ou nombres, symboles divers, signes diacritiques et signes de ponctuation.

Codage des caractères internationaux [4/5]

- La table complète devrait utiliser des mots de 32 bit (on par alors de UTF-32). En pratique, on utilise un sous-ensemble de cette table globale, codée sur 16 bits, appelée UTF-16.
- Enfin, pour des raisons de gains de place, il existe UTF-8 : les caractères de l'ASCII 7 bits sont codés sur 1 octets, les autres étant codés avec la syntaxe « **&#xCODE;** », avec CODE dans la table Unicode complète.
- Site web en français sur Unicode : http://hapax.qc.ca/

Codage des caractères internationaux [5/5]

- **Attention !!!** Avec l'UTF-8, tous les caractères n'utilisent pas tous la même place en mémoire. On n'a pas toujours « un caractère = un octet » ➔ attention aux débordements de pile…
- Dans la plupart des langages modernes, il existe des fonctions permettant de traduire des chaîne exprimées dans une page de code quelconque en Unicode (et réciproquement), de faire des comparaisons/concaténation de chaînes de caractères qui ne sont pas de l'ASCII simple, etc.

De l'intérêt du XML [1/8]

- Qu'avons nous définit :
  - Quelques règles syntaxiques de base d'un langage,
  - Langage dont les règles syntaxiques peuvent être définies à l'aide d'un fichier de paramétrage, exprimé généralement en DTD ou XML schéma ;
- Aussi, finalement, en quoi XML est-il intéressant (par exemple, lors des échanges de flux de données entre client et serveur) ?

De l'intérêt du XML [2/8]

- Qu'utilisait-on avant XML, pour s'échanger des données complexes ? Quelques exemples :
  - Les fichiers « positionnés ». Ex : caractères de 1 à 10 = numéro identifiant, de 11 à 40 = nom, de 41 à 70 = prénoms, etc. Critique :
    - Si le nom est de taille < 30 caractères, on utilise tout de même 30 caractères ;
    - Comment entrer un nom de taille > 30 caractères ?
    - Comment coder les zones optionnelles (ex : « 4 espaces » dans une zone de taille = 4 doivent ils être interprétés comme 4 espaces, ou comme une « zone non remplie » ?
    - Comment coder les items multivalués ?
    - Comment coder une arborescence (ex : structure contenant une structure) ?

De l'intérêt du XML [3/8]

- Autre solution classique : le fichier texte avec séparateur (ex : CSV, qui utilise le point virgule comme séparateur). Critique :
  - Et si l'item contient un caractère qui est le séparateur (ex : un « ; » dans un CSV) ?
  - Solution parfois employée : on met un caractère d'échappement devant le séparateur pour dire « le caractère qui suit n'est pas le séparateur ». Mais alors, comment transmettre ce caractère d'échappement ? En en mettant deux de suite ?
  - ainsi, on peux avoir besoin de plus de N octets pour coder N caractères (à cause de la possibilité d'avoir des caractères d'échappement) ➔ traitement plus complexe ➔ attention aux risques de dépassement de pile !!!
  - Même remarque que précédemment : comment coder les items multivalués, et les arborescences ?

De l'intérêt du XML [4/8]

- Critiques globales de ces « anciens formats » :
  - Mal adaptés aux modèles arborescents, et encore moins aux modèles objets,
  - Et surtout, l'explication d'un format se fait... en français (ou en anglais, ou tout autre langage naturel, bavard, peu formaliste, et surtout, soumis à interprétation)

De l'intérêt du XML [5/8]

- Or (Cf. exemple « Livre »), XML permet :
  - D'avoir des items multivalués (ex : auteur),
  - Peut décrire une arborescence,
  - Permet de contenir des paramètres optionnels,
  - D'imbriquer du code interprété avec un moteur externe (ex. des PI),
  - De définir le système de codage utilisé pour coder les caractères,
  - …

De l'intérêt du XML [6/8]

- Et si en plus, on ajoute le fait que :
  - Le XML schéma… c'est du XML, le XML est donc le langage :
    - Qui est utilisé pour transporter les données,
    - Et qui permet de définir la syntaxe des données transportées (utilisé par un parser),
  - Il existe un langage XML qui permet à un parser de traduire du XML vers un autre format, XML ou pas (exemple : XML vers XHTML, ou XML vers WML) : XSLT
  - Il existe des langages XML d'interrogation de données : Xpath (qui n'est pas vraiment du XML), et XML Query

De l'intérêt du XML [7/8]

- Alors :
  - Tout devient XML : le langage de description du langage qui transportera les données,
  - Le document qui contient les données,
  - Le document qui met en forme les données, ou qui les traduits en autre chose,
  - Les requêtes d'interrogation de données disponibles sous un format XML, etc.
- Conséquences :
  - langage souple, structurant, permettant d'interconnecter des équipements hétérogènes,
  - et existence de nombreux outils permettant de développer rapidement ET proprement.

De l'intérêt du XML [8/8]

- Peut-être quelques inconvénients (tout de même) :
  - XML est bavard (mais à l'époque des réseau Gbits/s…),
  - Ça fait beaucoup de langages XML à connaître.
- Il existe de nombreux schémas décrivant des syntaxes XML propres à chaque besoin :
  - OpenDocument (traitement de texte),
  - XMPP (messagerie instantanée Jabber),
  - VoiceXML (voix sur IP),
  - etc. (plusieurs centaines de formats normalisés)

DTD [1/7]

- Le DTD (Document Type Definition) définit la structure d'un document XML, les éléments, et les attributs autorisés, et le type de contenu ou d'attribut permis
- Le DTD n'est pas du XML (mais ∈ SGML)
- Le DTD peut être :
  - interne au document XML :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE Livres [
instructions de la DTD
]>
<Livres>[...]
```

  - externe (dans un fichier « `*.dtd` ») :

```
<!DOCTYPE Livres SYSTEM "URI">
```

DTD [2/7]

- Chaque instruction d'un DTD est introduite par : « **<!** »
- Chaque balise est définie par l'instruction « **<!ELEMENT** » suivie :
  - du nom que porte l'élément en question,
  - soit le nom des sous-éléments que peut contenir l'élément parent, soit le type de l'élément si celui-ci est un nœud enfant.

DTD [3/7]

- Chaque liste d'attributs d'une balise est définie par l'instruction « **<!ATTLIST** » :

```
<!ELEMENT nom_elmt (#PCDATA)>
<!ATTLIST nom_elmt
   attrib1 CDATA #REQUIRED "val_par_def"
   attrib2 CDATA #REQUIRED ...>
```

- Un attribut peut être :
  - **#REQUIRED** : obligatoire
  - **#IMPLIED** : c'est l'application qui décide quoi faire
  - **#FIXED** : imposé (on ne peut mettre autre chose)
- Mots clés utilisés dans les DTD :
  - **ANY** : Indique que l'élément défini peut avoir tout type de contenu légal. **<!ELEMENT nom_elmt ANY>**

DTD [4/7]

- **`EMPTY`** : Indique que l'élément défini ne peut pas avoir de contenu. **`<!ELEMENT nom_elmt EMPTY>`**
- **`(#PCDATA)`** : Indique que l'élément défini peut contenir une chaîne de caractères. Ce mot clé peut être associé à une liste de contenus; Dans ce cas, il se place en première position de la liste.
  `<!ELEMENT nom_elmt (#PCDATA)>`
- **`(..., ...)`** : Délimite une liste de contenus en précisant le nom et l'ordre des nœuds enfants.
  `<!ELEMENT nom_elmt_parent (nœud1, nœud2, nœud3, ...)>`

DTD [5/7]

- , : Opérateur de liaison « ET », indique que les nœuds enfants déclarés (dans une liste) sont tous obligatoires et dans l'ordre de leur déclaration.
  <!ELEMENT nom_elmt_parent (noeud1, noeud2, noeud3, ...)>
- | : Opérateur de liaison « OU », indique qu'un seul des nœuds enfants déclarés (dans une liste) peut être utilisé. C'est au choix.
  <!ELEMENT nom_elmt_parent (noeud1 | noeud2 | noeud3 | ...)>
- ? : Indique que le nœud enfant est optionnel. Si ce caractère est placé après les parenthèses d'une liste de contenu, il s'applique à tout le contenu.
  <!ELEMENT nom_elmt_parent (noeud1?, noeud2, noeud3, ...)>

DTD [6/7]

- `+` : Indique que le nœud enfant peut être utilisé une ou plusieurs fois. Si ce caractère est placé après les parenthèses d'une liste de contenu, il s'applique à tout le contenu.

```
<!ELEMENT nom_elmt_parent (noeud1 | noeud2 | noeud3 | ...)+>
```

- `*` : Indique que le nœud enfant est optionnel. Il peut être utilisé 0, 1 ou plusieurs fois. Si ce caractère est placé après les parenthèses d'une liste de contenu, il s'applique à tout le contenu.

```
<!ELEMENT nom_elmt_parent (#PCDATA | noeud1 | ...)*>
```

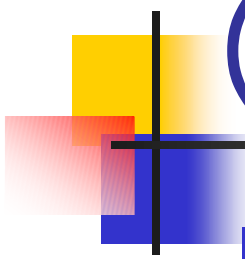
DTD [7/7]

- On peut imbriquer le parenthèsage. Ex :
```
<!ELEMENT nom_elmt_parent (fils1, fils2,
(fils3|fils3bis) ) >
```
- !!! En DTD, un élément contient des éléments, ou du texte CDATA, mais il n'est pas possible de mixer les deux. On ne peut définir en DTD un langage autorisant : **<B><I>bla</I> bla</B>**
- Le paramètre « **<!ENTITY** » permet de lier un DTD à un DTD externe :
```
<!ENTITY %ELEMENT_EXT SYSTEM "elm.dtd">
<!ELEMENT elmt (noeud1|%ELEMENT_EXT;)>
%ELEMENT_EXT;
```

TP1 : Créer un DTD

- Créer un DTD permettant de valider le document XML donné dans l'exemple <Livres>...</Livres> de la page 5 avec :
  - Il y a toujours au moins un livre dans livres,
  - Il y a toujours au moins un auteur,
  - Il y a toujours une édition et un titre,
  - Il peut y avoir 0, 1, ou N bibliographies,
  - Une édition contient toujours un éditeur, un ISBN, l'année de première impression étant optionnelle.

Initiation aux schémas XML (XSD) [1/6]



- Permet de définir un langage XML valide... en utilisant un langage XML
- Publié par le W3C en mai 2001
- Les schémas XML sont généralement stockés dans des fichiers « **`*.xsd`** » (XML Schema Definition)

Initiation aux schémas XML (XSD) [2/6]

- Comme tout document XML, un Schema XML commence par un prologue, et a un élément racine :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
<!-- déf éléments, attributs & types -->
</xsd:schema>
```

- L'élément racine est l'élément **`xsd:schema`**.

Initiation aux schémas XML (XSD) [3/6]

- Pour lier un élément à un schéma XML externe :

```
<?xml version="1.0" encoding="UTF-8"?>
<nom_element
xmlns:xsi="http://www.w3.org/2001/XMLSch
ema-instance"
xsi:noNamespaceSchemaLocation="nom_fichi
er.xsd">
    <elem2 attrib1="val" attrib2="val">
    <elem3 attrib3="val">
</nom_element>
```

Initiation aux schémas XML (XSD) [4/6]

- Déclarations d'éléments :
  - **`<xsd:element name="nom_elem" type="nom_type"></xsd:element>`**
  - « **`nom_type`** » peut être :
    - Un type prédéfini (string : **`xsd:string`**, date : **`xsd:date`**, etc.)
    - Un type définit par l'utilisateur
  - L'imbrications d'éléments est définie ainsi :

```
<xsd:element name="nom_elem">
 <xsd:complexeType>
  <xsd:sequence> ou <xsd:choice> ou <xsd:all>
   <xsd:element name="elem1">…</xsd:element>
   <xsd:element name="elem2">…</xsd:element>…
  </xsd:sequence> ou </xsd:choice> ou </xsd:all>
 </xsd:complexeType>
</xsd:element>
```

Initiation aux schémas XML (XSD) [5/6]

- Pour définir le nombre d'occurrences possibles d'un élément, on utilise les attributs **`minOccurs`** et **`maxOccurs`**. **`maxOccurs`** peut prendre comme valeur « **`unbounded`** » (illimité)
- Déclaration d'attributs :
  - Rq : ne peut contenir que des types simples
  - Syntaxe :

```
<xsd:attribute name="nom_attrib"
type="Nom_type" use="optional|required"
default="valeur_par_defaut" />
```

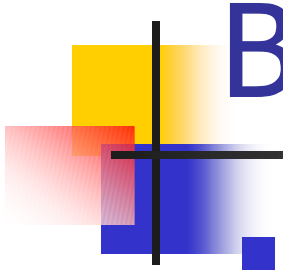
Initiation aux schémas XML (XSD) [6/6]

- Utilisation du mot clé « **ref** » :

```
<xsd:element name="elem1">
  …
</xsd:element>
<xsd:element name="elem2">
  …
</xsd:element>
<xsd:element name="elem3">
 <xsd:complexeType>
  <xsd:sequence>
   <xsd:element ref="elem1" />
   <xsd:element ref="elem2" />
  </xsd:sequence>
 </xsd:complexeType>
</xsd:element>
```

TP2 : Créer un schéma XML

- Même travail que le TP1, cette fois-ci en utilisant un schéma XML (à la place d'un DTD)



Bonus

- Voir le document « Transformations XSL (XSLT) » du W3C pour comprendre comment transformer un document XML valide en n'importe quoi
- Voir le document « Langage XML Path (XPath) » du W3C pour comprendre comment parcourir l'arborescence d'un document XML, afin de l'uriliser comme structure de données