1- Motivations

Nous avons vu comment il était possible d'écrire du XML en respectant ses règles de syntaxe et d'obtenir ainsi un document XML *bien formé*.

Nous allons maintenant décrire comment spécifier des contraintes plus précises et propres à notre langage XML. Cela va prendre ici la forme d'une DTD (Définition de Type de Document).

On dira alors qu'en plus d'être ***bien formé***, le document est ***valide*** par rapport à une certaine DTD.

Cette spécification d'une grammaire pour un langage et la possibilité de tester automatiquement son respect par un document donné, présentent les avantages suivants :

- faciliter l'échange et la mise en commun de documents produits par des rédacteurs différents ;
- aider les développeurs qui conçoivent des outils automatiques pour traiter les documents respectant la même DTD.

2- LIER UN FICHIER XML A UNE DTD

2.1- DTD INTERNE

Dans ce cas, la spécification de la DTD arrive dans l'entête du document XML : on trouve d'abord le mot-clef DOCTYPE suivi de l'élément servant de racine au document et enfin la DTD elle-même entre crochets :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE cv [
.
.
.
]>
<cv>
.
.
.
</cv>
```

2.2- DTD EXTERNE

Cette fois, la DTD est détachée dans un fichier séparé, on se contente d'y faire référence dans l'entête du document XML. On retrouve le mot-clef DOCTYPE suivi de l'élément servant de racine, puis le mot-clef SYSTEM suivi d'une URI menant au fichier DTD.

À noter également que la première ligne doit faire apparaître l'attribut standalone avec la valeur no.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no" ?>
<!DOCTYPE cv SYSTEM "cv.dtd">
<cv>
.
.
.
</cv>
```

À noter que, sans toucher au document XML, il est possible de faire le lien au moment de la validation. Par exemple, avec **xmllint**, on écrira :

xmllint --dtdvalid madtd.dtd mondoc.xml --noout

2.3- DTD MIXTE

Enfin, il est possible de mélanger les deux notations pour avoir une partie de la DTD dans un fichier séparé et une autre partie embarquée dans le document XML :

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no" ?>
<!DOCTYPE cv SYSTEM "cv.dtd" [
.
.
.
]>
<cv>
.
.
.
</cv>
```

3- DEFINIR LES ELEMENTS ET LEURS CONTENUS

Il s'agit ici de déclarer les éléments autorisés à apparaître dans le document, ainsi que leurs imbrications possibles. La forme générale est la suivante :

```
<!ELEMENT nom_element modèle_de_contenu>
```

Les noms des éléments (comme ceux des attributs) doivent être des *noms XML* :

- le premier caractère est une lettre quelconque ou un _ (underscore ou tiret bas) ;
- les caractères suivants peuvent être des lettres, des chiffres, des tirets bas (_), des traits d'union (-) ou des points (.) ;
- il n'y a pas de limitation sur la longueur d'un nom XML.

Nous passons maintenant en revue les différents modèles de contenu utilisables dans les DTD.

A- CONTENU PUREMENT TEXTUEL

Si l'élément peut contenir du texte brut mais pas de nouvelles balises, on utilisera le modèle de contenu PCDATA :

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

Aucune balise n'est donc tolérée dans ce type de contenu mais, par contre, il est possible d'y utiliser des entités.

A-1. SOUS-ELEMENTS

Ici, on va lister les sous-éléments pouvant apparaître dans le contenu, par exemple :

```
<!ELEMENT identité (prénom,nom)>
```

indique que l'élément identité doit contenir, *dans l'ordre*, un élément prénom, un élément nom, et rien d'autre.

Il est possible de moduler le nombre d'apparitions d'un sous-élément en utilisant des quantifieurs après les noms d'éléments. Les quantifieurs utilisables dans les DTD sont :

- ? : 0 ou 1 fois ;
- * : 0, 1 ou plus ;
- + : 1 ou plus.

L'exemple suivant indique que l'élément identité doit contenir, toujours en respectant l'ordre, un ou plusieurs éléments prénom, un élément surnom facultatif et exactement un élément nom :

```
<!ELEMENT identité (prénom+,surnom?,nom)>
```

A-2. Alternatives

Il est également possible de définir les sous-éléments qui peuvent apparaître de manière exclusive : si c'est l'un, ça n'est pas les autres. Dans l'exemple ci-dessous, une expérience professionnelle peut être soit un emploi, soit un stage :

```
<!ELEMENT expérience (stage|emploi)>
```

A-3. Combinaisons

Enfin, il est possible de combiner les syntaxes vues précédemment :

```
<!ELEMENT diplôme ( (intitulé,année) | (année,compétences,stage?)+ )>
```

Dans ce cas, un diplôme est :

- soit l'intitulé du diplôme et l'année d'obtention ;
- soit une suite d'année, avec les compétences acquises cette année là, éventuellement validées par un stage.

A-4. Contenu mixte

Une possibilité intéressante est de pouvoir mixer du texte brut avec des balises sans mettre plus de contraintes sur l'ordre et le nombre d'apparitions de ces balises. Cela se fait avec une alternative entre un contenu de type PCDATA et des sous-éléments, cette alternative pouvant se répéter plusieurs fois :

```
<!ELEMENT parcours (#PCDATA | diplôme)*>
```

Ici, on a un élément parcours qui a un contenu mixte : du texte pouvant contenir un nombre quelconque de sous-éléments diplôme.

A-5. Contenu vide

Un élément peut également n'avoir aucun contenu, comme c'est le cas par exemple de la balise br (retour à la ligne en HTML). Cela se précise dans la DTD de la manière suivante :

```
<!ELEMENT br EMPTY>
```

Une telle balise sera donc ouverte et immédiatement refermée avec la notation suivante :
.

A-6. CONTENU QUELCONQUE

On termine avec la possibilité d'autoriser n'importe quel contenu à apparaître dans un élément.

```
<!ELEMENT mavie ANY>
```

Dans ce cas, l'élément **mavie** pourra contenir du texte brut mélangé avec toute balise définie dans la DTD, dans n'importe quel ordre, autant de fois que l'on veut.

Ce modèle de contenu est en contradiction avec l'esprit des DTD, lesquelles visent plutôt à restreindre au maximum le rédacteur. Une utilisation cependant pratique du ANY intervient lors de la rédaction d'une DTD alors que les documents XML sont déjà existants : dans ce cas, on définit les éléments de plus haut niveau en indiquant ANY pour leurs contenus, si les documents sont valides, on reprend la DTD en affinant le contenu de chacun des éléments, etc.

A-7. UN EXEMPLE SIMPLE ET COMPLET

Une DTD que l'on enregistre dans un fichier nommé livres.dtd :

```
<!ELEMENT liste_livres (livre+)>
<!ELEMENT livre (titre,auteur+,éditeur,description?,prix)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT éditeur (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT prix (#PCDATA)>
```

et un fichier XML la respectant :

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no" ?>
<!DOCTYPE liste_livres SYSTEM "livres.dtd">
<liste_livres>
<livre>
<titre>Comprendre XSLT</titre>
<auteur>Bernd Amann</auteur>
<auteur>Philippe Rigaux</auteur>
<éditeur>O'Reilly</éditeur>
<description>
Le livre suit une double démarche de présentation des aspects les plus simples, puis, progressivement, les plus complexes du langage XSLT, et d'application des mécanismes de ce langage à des cas concrets d'utilisation. Il débute par un chapitre introductif en forme d'étude de cas qui propose, sur une application de type "Officiel des spectacles" adaptée au Web, une déclinaison des différents thèmes couverts.
</description>
<prix>33 euros</prix>
</livre>
<livre>
<titre>Learning XML</titre>
<auteur>Erik T. Ray</auteur>
<éditeur>O'Reilly</éditeur>
<prix>40 dollars</prix>
```

```
</livre>
</liste_livres>
```

4- DEFINIR LES ATTRIBUTS

La syntaxe des DTD utilise cette fois le mot **ATTLIST**, suivi du nom de l'élément concerné, suivi de la liste de ses attributs : pour chacun, on trouvera son nom, son type et son caractère optionnel ou non.

Comme nous l'avons déjà indiqué, les noms des attributs doivent être des *noms XML* :

- le premier caractère est une lettre quelconque ou un _ (underscore ou tiret bas) ;
- les caractères suivants peuvent être des lettres, des chiffres, des tirets bas (_), des traits d'union (-) ou des points (.) ;
- il n'y a pas de limitation sur la longueur d'un nom XML.

Une déclaration d'attributs typique aura la forme suivante :

```
<!ATTLIST identité prénom CDATA #REQUIRED
                  nom    CDATA #REQUIRED
                  surnom CDATA #IMPLIED>
```

Dans ce cas, l'élément identité possède trois attributs prénom, nom et surnom. Les deux premiers sont obligatoires (REQUIRED) et le dernier est optionnel (IMPLIED). À noter également la possibilité pour un attribut d'être FIXED, c'est-à-dire de prendre systématiquement la même valeur.

Nous passons maintenant en revue les différents types d'attributs.

4.1- CDATA

Type général qui permet de saisir un texte quelconque pour un attribut de ce type.

A- NMTOKEN

Il s'agit ici un nom XML mais sans restriction sur le premier caractère (qui peut donc être un chiffre). Une contrainte essentielle est donc qu'un attribut de ce type ne contiendra pas d'espace.

Par exemple, un code postal pourra être déclaré de ce type :

```
<!ATTLIST ville nom  CDATA   #REQUIRED
                code NMTOKEN #REQUIRED>
```

B- NMTOKENS

Une suite de NMTOKEN, séparés par des espaces.

C- ÉNUMERATION

Dans ce cas, l'attribut ne peut prendre qu'un nombre fini de valeurs et l'on en donne la liste exhaustive, ces valeurs étant séparées par des |.

```
<!ATTLIST date jour (lundi|mardi|mercredi|jeudi|vendredi|samedi|dimanche) #REQUIRED
        num   NMTOKEN #REQUIRED
        mois  NMTOKEN #REQUIRED
        année NMTOKEN #REQUIRED>
```

D- ID

Il doit s'agir d'un *nom XML* qui identifie de manière unique l'élément. Autrement dit, une valeur qui apparaît dans un tel attribut ne peut pas apparaître une seconde fois dans le même document.

```
<!ATTLIST diplôme intitulé CDATA #REQUIRED
          code     ID    #REQUIRED>
```

Cette notion est tout à fait comparable à la clef primaire d'une table dans une base de données. Cependant, il faut garder à l'esprit qu'un ID ne peut pas être un nombre (car la valeur d'un ID doit être un *nom XML*).

E- IDREF

Un attribut de type IDREF doit contenir une valeur utilisée comme ID ailleurs dans le document.

```
<!ATTLIST stage entreprise CDATA #REQUIRED
          diplôme    IDREF #REQUIRED>
```

F- IDREFS

Une suite de IDREF, séparées par des espaces.

G- ENTITY

La valeur d'un attribut de ce type doit être une entité définie dans la DTD (voir section suivante pour la définition des entités).

H- ENTITIES

Une suite de ENTITY, séparées par des espaces.

5- DEFINIR LES ENTITES

Intuitivement, il s'agit de définir des raccourcis (ou des alias) qui seront utilisables dans les documents XML liés à la DTD. Certaines entités sont déjà définies en XML : < (<), > (>), & (&), " ("), et ' (').

5.1- ENTITES GENERALES

Le mot ENTITY suivi du nom de l'entité, puis de sa valeur entre guillemets ou apostrophes.

```
<!ENTITY ac "anticonstitutionnellement">
```

Cette déclaration permettra d'utiliser ∾ dans le document XML associé à la DTD.

La valeur de l'entité peut contenir du texte, des balises et des entités (le code XML ainsi défini doit être bien formé). Par exemple,

```
<!ENTITY piedpage '<hr />

<p>

Copyright 2005, dernière mise à jour octobre 2005.

</p>'>
```

5.2- ENTITES GENERALES EXTERNES

Si le code associé à une entité devient très important, il peut être intéressant de le détacher dans un fichier à part, ce que permet la syntaxe suivante :

```
<!ENTITY piedpage SYSTEM "pp.xml">
```

Dans ce cas, le fichier pp.xml doit être un fichier XML bien formé (première ligne de déclaration de la version XML et du codage utilisé, fermeture systématique des balises ouvertes, etc.).

5.3- ENTITES GENERALES EXTERNES NON PARSEES

Enfin, il y a le cas où le fichier externe représenté par l'entité ne contient pas du XML et ne doit donc pas être parcouru par les applications traitant le document.

```
<!NOTATION jpeg SYSTEM "image/jpeg">
<!ENTITY maphoto SYSTEM "toto.jpg" NDATA jpeg>
```

Ici, en plus du fichier externe, on trouve NDATA pour *Notation Data* et jpeg faisant référence à une NOTATION définie précédemment. La NOTATION doit permettre à l'application de traiter le fichier externe : il peut s'agir d'un type MIME comme ici ou d'une commande à exécuter.

Une telle entité pourra apparaître comme valeur d'un attribut défini avec le type ENTITY.

5.3- INTEGRER LES ENTITES

Naturellement, les entités définies dans la DTD et utilisées dans le document XML devront tôt ou tard être remplacées par leurs véritables valeurs

Certains navigateurs opèrent ce remplacement lorsque l'on les utilise pour visualiser ce document XML. Ca n'est pas le cas de Firefox par exemple qui choisit de ne pas remplacer les entités et cela pour des raisons de sécurité.

Un moyen de tester les entités est d'utiliser xmllint :

```
xmllint doc.xml --valid --noent
```

on charge la dtd avec --valid et on demande la substitution des entités avec --noent.

5.4- ENTITES PARAMETRES

Ce type d'entité permet d'éviter de répéter les mêmes informations. Par exemple, on donnera toujours le nom et le prénom d'une personne, quel que soit son statut dans le document :

```
<!ELEMENT identité (nom,prénom,naissance)>
<!ELEMENT enseignant (nom,prénom)>
```

L'utilisation d'une entité paramètre comme suit permet de ne pas répéter et autorise à enrichir ultérieurement la description d'une personne :

```
<!ENTITY % elements_personne "nom,prénom">

<!ELEMENT identité   (%elements_personne;,naissance)>

<!ELEMENT enseignant (%elements_personne)>
```

6- DTD MODULAIRES

Des instructions (INCLUDE et IGNORE) permettent de prendre en compte un bloc de la DTD, ou de l'ignorer. La syntaxe est la suivante :

```
<![IGNORE[
<!ELEMENT personne (nom,prénom)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prénom (#PCDATA)>
]]>


<![INCLUDE[
<!ELEMENT entreprise (dénomination,taille)>
<!ELEMENT dénomination (#PCDATA)>
<!ELEMENT taille (#PCDATA)>
]]>
```

Ce mécanisme devient puissant avec l'utilisation conjointe d'entités-paramètres.

```
<!ENTITY % bloc_personnes "INCLUDE">
<!ENTITY % bloc_entreprises "IGNORE">

<![%bloc_personnes;[
<!ELEMENT personne (nom,prénom)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prénom (#PCDATA)>
]]>

<![%bloc_entreprises[
<!ELEMENT entreprise (dénomination,taille)>
<!ELEMENT dénomination (#PCDATA)>
```

```
<!ELEMENT taille (#PCDATA)>
]]>
```

Il est notable que ce choix ait été fait par le W3C pour définir la DTD du XHTML.

7- BILAN SUR LES DTD

Les reproches suivants sont systématiquement faits aux DTD :

- l'élément racine n'est pas spécifié pas dans la DTD ; un document peut être valide en utilisant n'importe quelle balise de la DTD comme racine ;
- le nombre d'apparitions d'un élément ne peut pas être contraint précisément, puisque l'on ne dispose que des quantifieurs ?, * et +, on aimerait pouvoir dire qu'un élément doit apparaître plus de 2 fois mais toujours moins de 5 ;
- on ne dispose pas de types pour les contenus des attributs et des éléments (nom, date, code postal, numéro de téléphone, url, adresse mail, etc.) ;
- on ne peut pas contraindre la forme de ces contenus (entre 5 et 20 caractères, contenant un signe @, etc.) ;
- enfin, le langage utilisé pour définir une DTD n'est pas un langage XML !

Pour palier à ces manques, d'autres propositions ont été faites, permettant elles aussi de spécifier un langage XML, citons les *XML Schema* et *Relax NG*.

Série de TD 3 - DTD

Exo1- Soit la DTD suivante qui permet d'écrire des documents répertoires d'adresses :

```
<!ELEMENT répertoire (catégorie*)>
 <!ELEMENT catégorie  (contact*) >
 <!ATTLIST catégorie nom ID #REQUIRED>
 <!ELEMENT contact (bureau, maison?) >
 <!ATTLIST contact
        nom ID #REQUIRED
        prénom CDATA #REQUIRED >
 <!ELEMENT bureau EMPTY >
<!ATTLIST bureau
        téléphone CDATA #REQUIRED
        adresse CDATA #REQUIRED
        courriel CDATA #IMPLIED>
 <!ELEMENT maison EMPTY >
 <!ATTLIST maison
        téléphone CDATA #REQUIRED
        adresse CDATA #REQUIRED
        courriel CDATA #IMPLIED>
```

Soit le document ci-dessous. Il présente des erreurs de syntaxe et des erreurs de validité.

```
01 :<?XML version="1.0" encodage="UTF-8"?>
02 :<!DOCTYPE répertoire SYSTEM "repertoire.dtd">
03 :<repertoire>
04 :  <catégorie nom="1-perso" >
05 :    <contact nom="desvignes" prénom="nicole">
06 :      <bureau téléphone="0234542312" adresse="2 rue de l'espoir 44566 Chignole"/>
07 :      <maison téléphone="0870754566" adresse="2 rue de l'espoir 44566 Chignole"/>
08 :    </contact>
09 :    <contact nom="fonteau" prénom="jérémie">
10 :      <maison téléphone="0445879044" adresse="6bis avenue Jean Charcot 65322 St Mazan"/>
11 :      <bureau téléphone="0465345622" adresse="54 bd de la liberté 65444 jalibert">
12 :        Batiment 6, Porte 5, Bureau 304
13 :      </bureau></contact>
14 :  </catégorie>
15 :  <catégorie nom="2-travail">
16 :    <contact nom="martin" prénom="paul">
17 :      <bureau téléphone=0245664442 adresse="5 rue de l'épine 45665 Astorie"
18 :                              courriel="paul.martin@gmail.com" fax="0245664412"/>
19 :    </contact>
20 :    <contact nom="batiset" prénom="raymond">
21 :      <maison téléphone="0248666445" adresse="5 rue de l'épine 45665 Astorie"
22 :                              courriel="rbatisset@free.fr"/>
23 :    <contact nom="fonteau" prénom="paul">
24 :      <bureau téléphone="0245678966" adresse="34 chemin de la fraise 56789 Ghéno"
25 :                              email="paul-fonteau@yahoo.fr"/>
26 :      <maison téléphone="0256780032" adresse="11 rue du petit plaisir 56888 Trigonet"/>
27 :    </contact>
28 :  </catégorie>
29 :</repertoire>
```

Question : Rechercher les erreurs. Pour chaque erreur indiquer la ligne où elle se trouve, sa nature (syntaxe ou validité), la correction à apporter pour que le document soit bien formé et valide.

Exo2 :

```
<!-- début de la DTD -->
<!DOCTYPE textemath [
<!-- DTD pour décrire un texte contenant des formules mathématiques -->
```

```
<!ELEMENT textemath ((texte | formule)+) >
<!ELEMENT texte (#PCDATA) >
<!ELEMENT formule (valeur|somme|difference|produit|fraction|racine|puissance) >
<!ELEMENT valeur (#PCDATA)>
<!ELEMENT somme (op1, op2)>
<!ELEMENT difference (op1, op2)>
<!ELEMENT produit (op1, op2)>
<!ELEMENT fraction (op1, op2)>
<!ELEMENT racine (op1)>
<!ATTLIST racine ordre CDATA #IMPLIED>
<!ELEMENT puissance (op1)
<!ATTLIST puissance exposant CDATA #REQUIRED>
<!ELEMENT op1 (valeur|formule)>
<!ELEMENT op2 (valeur|formule)>
]>
<!-- fin de la DTD -->
```

Question : Écrire un document XML valide (conforme à la DTD ci-dessus) destiné à transmettre l'énoncé suivant :

Calculer la valeur de l'expression $\frac{x^4+\sqrt[3]{5}}{7\sqrt{3}}$ lorsque x prend la valeur 4 .

Solution

Exo1 :

Solutions des exercices

Solution exercice 1

1) Rechercher les erreurs. Pour chaque erreur indiquer la ligne où elle se trouve, sa nature (syntaxe ou validité), la correction à apporter pour que le document soit bien formé et valide.

Ligne	Nature erreur	Description erreur et correction
1	syntaxe	XML doit être remplacé par xml en minuscules
1	syntaxe	"encodage" n'existe pas, à remplacer par "encoding"
4 et 15	syntaxe	les valeurs de l'attribut "nom" ne sont pas des "noms XML"
17	syntaxe	il manque des séparateurs autour du numéro de téléphone
entre 22 et 23	syntaxe	il manque la balise fermante de l'élément "contact"
3 et 29	validité	l'élément s'appelle "répertoire", avec un accent, et non "repertoire"
10 et 11	Validité	l'élément bureau doit précéder l'élément "maison"
12	validité	la balise "bureau" doit être vide
entre 20 et 21	validité	il manque un élément "bureau"
18	validité	l'attribut "fax" n'existe pas"
23	validité	la valeur de l'attribut "nom" doit être unique dans le document (type ID)
25	validité	l'attribut "email" n'existe pas

Solution exercice 2

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- début de la DTD -->
<!DOCTYPE textemath SYSTEM "math.dtd">
<!-- fin de la DTD -->
<textemath>
  <texte>Calculer la valeur de l'expression</texte>
  <formule>
    <fraction>
     <op1>
       <formule>
         <somme>
          <op1>
            <formule>
              <puissance exposant = "4"><op1><variable>x</variable></op1></puissance>
            </formule>
          </op1>
          <op2>
            <formule>
              <racine ordre= "3"><op1><valeur>5</valeur></op1></racine>
            </formule>
          </op2>
         </somme>
       </formule>
     </op1>
     <op2>
       <formule>
         <produit>
          <op1><valeur>2</valeur></op1>
          <op2>
            <formule>
              <racine><op1><valeur>3</valeur></op1></racine>
            </formule>
          </op2>
         </produit>
       </formule>
     </op2>
   </fraction>
  </formule>
  <texte>lorsque</texte>
  <formule><variable>x</variable></formule>
  <texte>prend la valeur</texte>
  <formule><valeur>4</valeur></formule>
</textemath>
```