Décrire et manipuler un document numérique

Cours 7 : DTD et modélisation

Loïc Grobol <lgrobol@parisnanterre.fr>

2024-03-04

Document bien formé

On a vu que XML était à la fois plus souple et plus strict que HTML

- Plus souple : on peut définir les types d'éléments et d'attributs qu'on veut.
- Plus strict : la syntaxe est moins permissive.
  - Ou plutôt : les interpréteurs n'essaient pas de deviner en cas d'erreur de syntaxe.

Ce cours est inspiré entre autres des transparents XML et données internet de Pierre Nerzic

---

Les règles de syntaxe qu'on a vu dans le cours précédent définissent le fait d'être **bien formé** :

- Le document commence par une déclaration XML.
- Le document a un unique élément racine.
- Les éléments et attributs ont des noms autorisés.
- Les éléments ont une balise de fermeture ou sont autofermants.
- Les éléments sont positionnés correctement (pas d'imbrication sans inclusion).
- Les valeurs d'attribut XML sont entre simple `'` ou double `"` quotes.
- Les entités sont utilisées pour les caractères spéciaux.

Pour avoir une définition complète, mais très, très formelle : voir la spécification.

Document valide

Même si un document XML est bien formé, ça ne signifie pas qu'il soit approprié pour une application.

Imaginez par exemple qu'un système de bibliographie soit prévu pour des données de la forme suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<bibliography>
<entry type="journalArticle">
  <title>The Misgendering Machines: Trans/HCI Implications of Automatic Gender Recognition</t
</entry>
<entry type="book">
  <title>Introduction aux études sur le genre</title>
</entry>
</bibliography>
```

Si on lui donne l'entrée suivante, qu'en faire ?

```
<?xml version="1.0" encoding="UTF-8"?>
<bibliography>
<book>
  <title>Gender Trouble: Feminism and the Subversion of Identity</title>
</book>
</bibliography>
```

Le document est bien formé, mais il ne correspond pas aux attentes : il n'est pas **valide**

(Le terme n'est pas très heureux)

On dit qu'un document est valide s'il est conforme à une spécification qui décrit

- Les types d'éléments autorisés.
- Les attributs autorisés pour chaque type d'élément.
- Les usages autorisés pour les éléments, et notamment les structures.

Il existe plusieurs standard pour décrire une telle spécification : DTD, XML Schemas, RelaxNG, Schematron…

Ces langages modélisent des règles de validité plus ou moins précises et d'une manière plus ou moins lisible.

Valider un fichier XML, c'est le comparer à cette spécification à l'aide d'un outil qui indique

- Soit le document est valide, conforme aux règles,
- Soit il contient des erreurs comme : tel attribut de tel élément contient une valeur interdite par telle contrainte, il manque tel sous-élément ou attribut dans tel élément…

Dans ce cours, on se limitera aux DTD, plus anciennes et plus simples.

DTD

Une *Document Type Definitions* est une liste de règles définies au début d'un document XML pour permettre sa validation pendant sa lecture. Elle est déclarée par un élément spécial `DOCTYPE` juste après le prologue et avant la racine :

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE itineraire ... >
<itineraire nom="essai">
<etape distance="0km">départ</etape>
<etape distance="1km">tourner à droite</etape>
</itineraire>
```

Attention : la syntaxe n'est pas la même que celle du contenu du XML

DTD interne ou externe

Une DTD peut être

- **Interne**, intégrée au document, entre `[ ]` :

  ```
  <!DOCTYPE itineraire [
  ...
  ]>
  ```

- **Externe**, dans un autre fichier, signalé par `SYSTEM` suivi de l'URL du fichier :

```
<!DOCTYPE itineraire SYSTEM "itineraire.dtd">
```

- **Mixte**, il y a à la fois un fichier et des définitions locales :

```
<!DOCTYPE itineraire SYSTEM "itineraire.dtd" [
...
]>
```

Une DTD externe peut être utilisée pour plusieurs fichiers XML, c'est donc ce qu'on utilise pour la plupart des applications.

Fichiers *standalone*

Vous trouverez parfois un attribut `standalone` valant `"yes"` ou `"no"` dans les prologues XML. Il est optionnel et présent uniquement quand il y a une DTD interne.

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

- `"no"` (par défaut) : la DTD interne fournit les valeurs par défaut et les entités et il peut y avoir des définitions externes (`SYSTEM`).
- `"yes"` : la DTD interne ne sert que pour la validation et ne peut pas employer d'entités ou de règles externes. Le document XML et avec sa DTD interne est totalement indépendant.

Validateurs

De nombreux outils de validation existent, souvent associés à des éditeurs XML. Dans le cadre de ce cours, on peut utiliser l'extension XML de vscode ou se contenter d'un validateur simple comme https://www.truugo.com/xml_validator/.

Structure d'une DTD

Une DTD contient des règles comme celles-ci :

```
<!ELEMENT itineraire (etape+)>
<!ATTLIST itineraire nom CDATA #IMPLIED>
<!ELEMENT etape (#PCDATA)>
<!ATTLIST etape distance CDATA #REQUIRED>
```

Ce sont des règles qui définissent :

- Des éléments (`ELEMENT`) : leur nom et le contenu autorisé,
- Des attributs (`ATTLIST`) : leur nom et options.

Le nom présent après le mot-clé `DOCTYPE` indique la racine du document. C'est un élément qui est défini dans la DTD.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!DOCTYPE itineraire [
<!ELEMENT itineraire (etape+)>
<!ATTLIST itineraire nom CDATA #IMPLIED>
<!ELEMENT etape (#PCDATA)>
<!ATTLIST etape distance CDATA #REQUIRED>
]>
<itineraire nom="essai">
<etape distance="0km">départ</etape>
<etape distance="1km">tourner à droite</etape>
</itineraire>
```

`<!DOCTYPE itineraire` définit la racine `<itineraire>`.

Éléments

La règle `<!ELEMENT nom contenu>` définit un élément : son nom et ce qu'il peut y avoir entre ses balises ouvrantes et fermantes.

La définition du contenu peut prendre différentes formes :

- `EMPTY` : signifie que l'élément doit être vide,
- `ANY` : signifie que l'élément peut contenir n'importe quels éléments (définis dans la DTD) et textes (leur ordre d'apparition et leur nombre ne seront pas testés),
- (`#PCDATA`) : signifie que l'élément ne contient que du texte,
- (définitions de sous-éléments) : spécifie les sous-éléments qui peuvent être employés.

Voici un exemple d'éléments simples :

```xml
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!DOCTYPE itineraire [
  <!ELEMENT itineraire ANY>
  <!ELEMENT boucle EMPTY>
  <!ELEMENT etape (#PCDATA)>
]>
<itineraire>
  <boucle/>
  <etape>départ</etape>
  <etape>tourner à droite</etape>
  <boucle></boucle>
  <itineraire/> <!-- c'est possible avec ANY -->
  <etape>tourner à gauche</etape>
</itineraire>
```

On peut restreindre les sous-éléments autorisés en donnant une liste ordonnée dans laquelle chaque sous-élément peut être suivi d'un quantificateur parmi `*`, `+` et `?` pour indiquer des répétitions, comme dans les expressions régulières.

```xml
<!ELEMENT itineraire (boucle?, etape+, variante*)>
```

Se lit ainsi : l'élément `<boucle>` est en option, il doit être suivi d'au moins un `<etape>` puis de zéro ou plus `<variante>`.

La liste peut aussi contenir des alternatives notées `( contenu1 | contenu2 | … )` :

```xml
<!ELEMENT informations (topoguide | carte)>
```

signifie que l'élément `<information>` peut contenir soit un enfant `<topoguide>`, soit un enfant `<carte>`.

On peut grouper plusieurs séquences avec des parenthèses pour spécifier ce qu'on désire :

```
<!ELEMENT personne (titre?, (nom,prenom+) | (prenom+,nom))>
```

On peut aussi indiquer que l'élément peut contenir du texte ou des sous-éléments :

```
<!ELEMENT etape (#PCDATA | waypoint)* >
```

```
<etape>avancer tout droit</etape>
<etape><waypoint lon="3.1" lat="48.2"/></etape>
<etape><waypoint lon="3.2" lat="48.1"/>aller au phare</etape>
```

Attributs

On décrit un attribut avec une règle de la forme `<!ATTLIST elem attr type valeur ...>` :

- Le nom de l'élément concerné
- Le nom de l'attribut,
- Son type
- Sa valeur.
  - `#REQUIRED` en tant que valeur indique que l'attribut est obligatoire
  - `#IMPLIED` qu'il est optionnel,
  - Si on fournit une chaîne `"…"`, c'est la valeur par défaut.

```
<!ELEMENT waypoint EMPTY>
<!ATTLIST waypoint
  lon CDATA #REQUIRED
  lat CDATA #REQUIRED
  ele CDATA #IMPLIED
  precision CDATA "50m"
  source (gps|utilisateur|carte) "carte">
```

Types d'attributs

Il y a plusieurs types possibles, voici les plus utiles :

- `CDATA` l'attribut peut prendre n'importe quelle valeur texte. Ne pas confondre avec `#PCDATA`.
- `(mot1|mot2|...)` Cela force l'attribut à avoir l'une des valeurs de l'énumération.
- `ID` l'attribut est un **identifiant** XML
  - Sa valeur doit être une chaîne (pas un nombre) unique parmi tous les autres attributs de type ID du document.
  - Il ne peut y avoir qu'un seul attribut de type `ID` par élément.
- `IDREF` l'attribut doit être égal, dans le document XML, à l'identifiant d'un autre élément.

Entités

Les entités sont des symboles qui représentent des morceaux d'arbre XML ou des textes.

```xml
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!DOCTYPE itineraire [
  <!ELEMENT auteur ANY>
  <!ELEMENT etape ANY>
  <!ENTITY copyright "© IUT Lannion 2022">
  <!ENTITY depart "<etape>Point de départ</etape>">
  <!ENTITY equipement SYSTEM "equipement.xml">
]>
<itineraire>
  <auteur>&copyright;</auteur>
  &equipement;
  &depart;
</itineraire>
```

```xml
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<!DOCTYPE itineraire [
  <!ELEMENT auteur ANY>
  <!ELEMENT etape ANY>
  <!ENTITY copyright "© IUT Lannion 2022">
```

```
  <!ENTITY depart "<etape>Point de départ</etape>">
  <!ENTITY equipement SYSTEM "equipement.xml">
]>
<itineraire>
  <auteur>© IUT Lannion 2022</auteur>
  [Le contenu du fichier equipement.xml]
  <etape>Point de départ</etape>
</itineraire>
```

Attention une entité ne peut remplacer que du XML bien formé et complet ! Ceci n'est pas valide :

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
  <!DOCTYPE itineraire [
  <!ELEMENT auteur ANY>
  <!ELEMENT etape ANY>
  <!ENTITY depart "<etape>Point de départ">
]>
<itineraire>
  &depart;</etape>
</itineraire>
```

Pour que ça le soit, c'est dans la définition de l'entité `depart` qu'il faut mettre `</etape>`