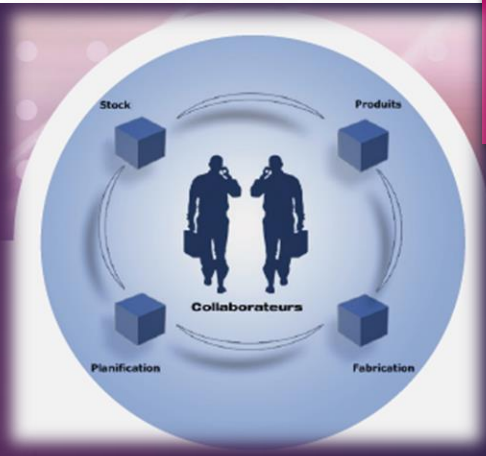
Université **d**e **B**ATNA 2
Département d'informatique



Les données semi-structurées

L3 ISIL

« Les données semi-structurées »
Chapitre 3 : La validation d'un document XML

Djennane A,
djennane.am@gmail.com

2020-2021

Unifier la structure d'un document XML,

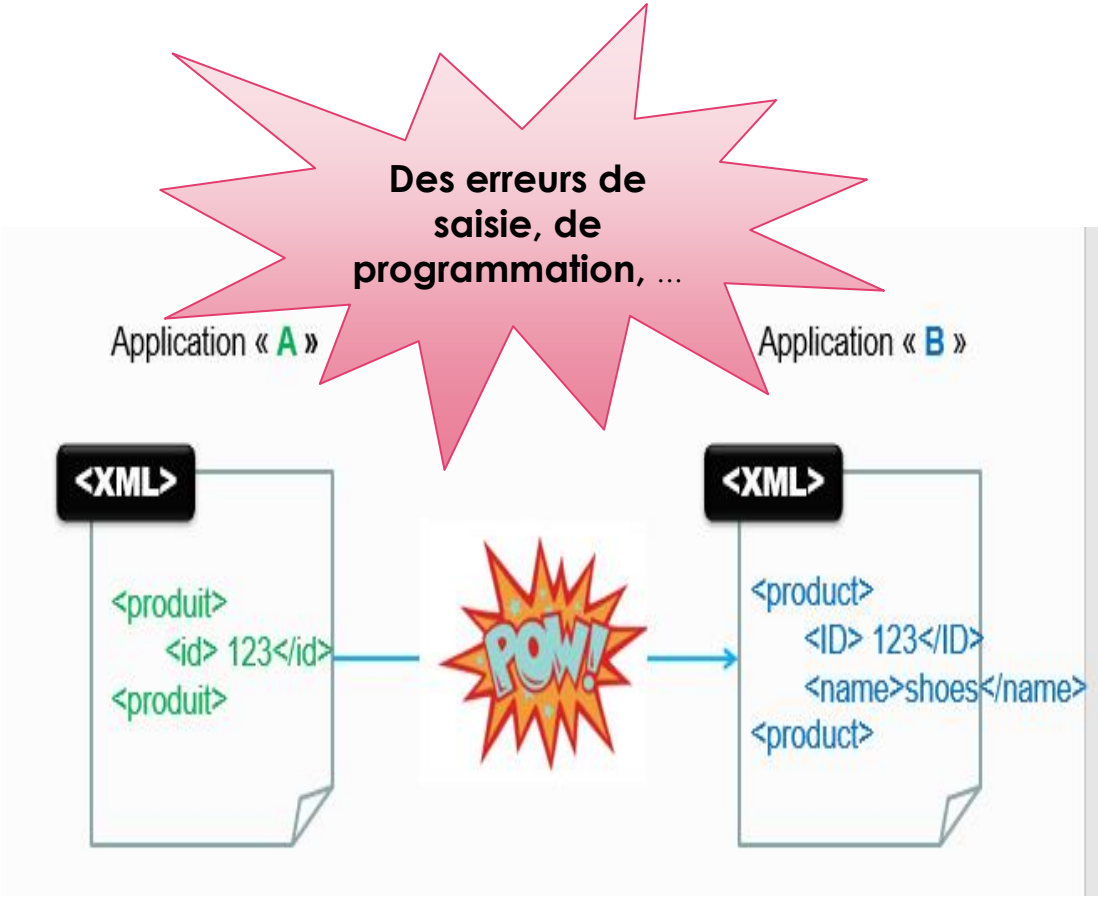
Une application a besoins d'un doc XML écrit par un autre programmeurs, ou généré par une application (données).

Prob :

- Comment définir une structure uniforme suivie par tout le monde pour assurer la cohérence ?
- Comment valider ce document ?
- Comment automatiser l'exploitation du document par d'autres applications ?

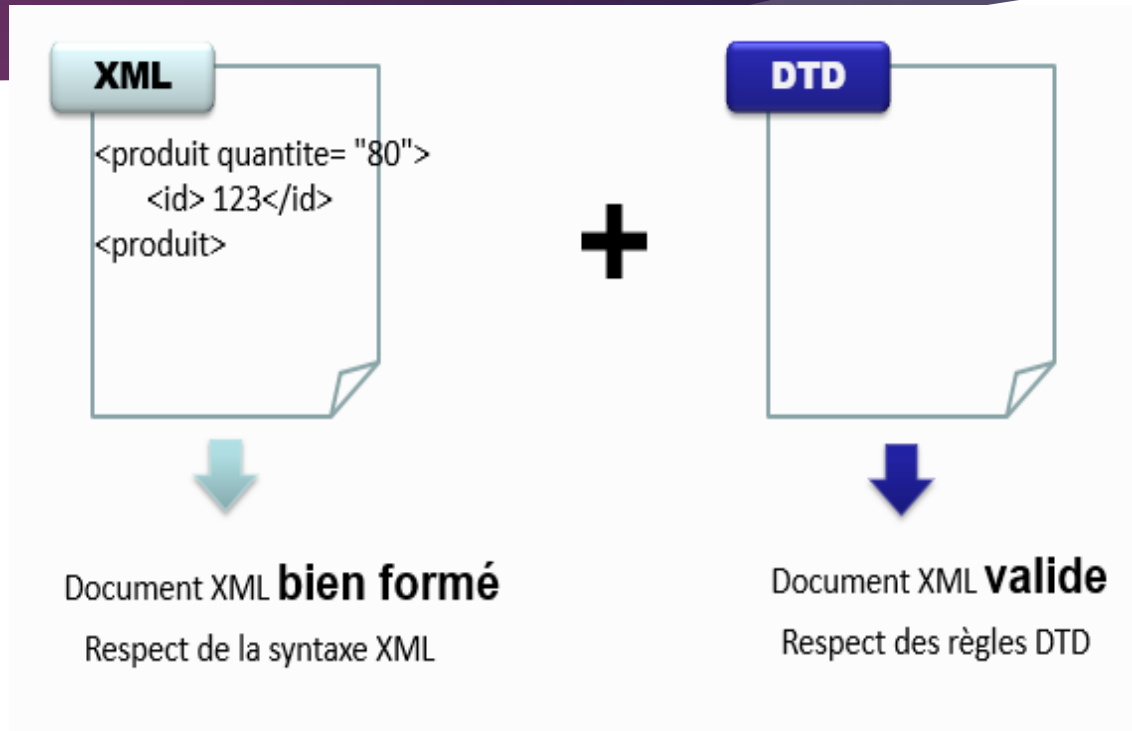
Solution :

- Des **règles** et des **contraintes** à imposer à l'édition qui facilitera la **validation** lors de l'exploitation du document XML,



Rôle de la validation dans l'entreprise

- **Un document Valide** =>
  - **Bien formé** (correcte syntaxiquement), mais pas nécessairement **valide**
  - **Conforme à une grammaire** (DTD), ou au schéma (XML scheme).



La validation va renforcer la qualité des échanges en contraignant l'émetteur de données et le consommateur de données à vérifier la cohérence des données structurées en XML.

La validation par DTD (*Document type definition*)

La plupart des outils, et notamment les parseurs XML, proposent des outils de validation. Les parseurs courants supportent une ou plusieurs formes de grammaires. Les DTD (*Document Type Definition*), étant la forme la plus ancienne, sont présentes dans la plupart des outils. Viennent ensuite ce qu'on nomme les schémas W3C, une forme de grammaire plus moderne mais également plus complexe. Enfin, il existe d'autres alternatives dont l'avenir est encore incertain, même si certains développeurs semblent déjà conquis par la simplicité de RelaxNG, par exemple.

DTD (*Document Type Definition*)

- Un DTD est un grammaire (ensemble de règles).
- Permet d'assurer la cohérence d'un document XML.

Par cohérence, il faut entendre à la fois le vocabulaire (éléments, attributs et espaces de noms) mais également, chose aussi importante, l'ordre et les quantités. En fin de compte, la validation revient à établir un visa sur le document XML.

Les formes d'écriture des DTD

- Une DTD peut être **interne** (partie du document XML) ou **externe** (un fichier) au document XML.
- La forme **externe** est plus **préférable** pour des raisons de **maintenance** et de **facilité d'accès**.

Les DTD **externes** peuvent être séparées en deux catégories :

1

- Privée (utilisant le mot clé SYSTEM) : fichier accessible uniquement en local
- publique (utilisant le mot clé PUBLIC) : disponible pour tout le monde

Exemple DTD interne

- DTD interne

```
<!DOCTYPE root-element [ declarations ]>
```

- Exemple

```
<?xml version="1.0" standalone="yes"?>

<!DOCTYPE parent [
<!ELEMENT parent (garcon,fille)>
<!ELEMENT garcon (#PCDATA)>
<!ELEMENT fille (#PCDATA)>
]>
<parent>
   <garcon>Loic</garcon>
   <fille>Marine</fille>
</parent>
```

DTD externe de type SYSTEM :

- Le fichier parent.xml

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE parent SYSTEM "parent.dtd">
<parent>
   <garcon>Loic</garcon>
   <fille>Marine</fille>
</parent>
```

- Le fichier parent.dtd contient :

```
<!ELEMENT parent (garcon,fille)>
<!ELEMENT garcon (#PCDATA)>
<!ELEMENT fille (#PCDATA)>
```

DTD

- DTD externe de type PUBLIC :
  - Exemple : référence à la DTD XHTML
    - <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

Syntaxe de DTD : Déclaration d'éléments

- Définition
  - Chaque élément du document doit être défini par une commande du type

    <!ELEMENT nom (contenu) >
  - où nom est le nom de l'élément (balise) et
  - où contenu décrit :
    - soit la structure de l'élément s'il est composé
    - soit #PCDATA si c'est une feuille
- Exemple

  <!ELEMENT livre (auteur, éditeur)>
  - définit un élément livre composé d'une séquence d'éléments auteur et éditeur

Élément vide

- Un élément vide est un élément qui n'a aucun contenu
- Déclaration :

```
<!ELEMENT elem-vide EMPTY>
```

Forme du contenu

Notations

- (a, b) séquence
- (a|b) liste de choix
- a? élément optionnel [0,1]
- a* élément répétitif [0,N]
- a+ élément répétitif [1,N]

Exemples

- (nom, prenom, rue, ville)
- (oui|non)
- (nom, prenom?, rue, ville)
- (produit*, client)
- (produit*, vendeur+)

Voici le détail de ces opérateurs :

- `*` : 0 à n fois ;
- `+` : 1 à n fois ;
- `?` : 0 ou 1 fois.

Quelques exemples :

```
<!ELEMENT plan (introduction?,chapitre+,conclusion?)>
```

L'élément `plan` contient un élément `introduction` optionnel, suivi d'au moins un élément `chapitre` et se termine par un élément `conclusion` optionnel également.

```
<!ELEMENT chapitre (auteur*,paragraphe+)>
```

L'élément `chapitre` contient de 0 à n éléments `auteur` suivi d'au moins un élément `paragraphe`.

```
<!ELEMENT livre (auteur?,chapitre)+>
```

L'élément `livre` contient au moins un élément, chaque élément, étant un groupe d'éléments où l'élément `auteur`, est optionnel et l'élément `chapitre` est présent en un seul exemplaire.

- Élément à contenu mixte
  - Contient du texte et des éléments
- Exemple de déclaration :

```
<!ELEMENT bonjour (#PCDATA | cible)*>
```

- Exemple d'utilisation :

```
<bonjour>
  Hello
  <cible>World</cible>
</bonjour>
```

Définition des attributs

Définition

<!ATTLIST tag [attribut type #mode [valeur]]* >

- Définit la liste d'attributs pour une balise, comme par exemple les attributs genre et ville pour la balise auteur, et l'attribut ville pour la balise éditeur :

```
<!ATTLIST auteur genre CDATA #REQUIRED
                 ville CDATA #IMPLIED>

<!ATTLIST editeur ville CDATA #FIXED "Paris">
```

REQUIRED = Obligatoire

IMPLIED = Optionnel

FIXED = valeur fixe

Lors du traitement par le parseur, une violation de ces règles provoquera une erreur

Le type des attributs

Le `TYPE` peut être principalement :

- `CDATA` : du texte (*Character Data*) ;
- `ID` : un identifiant unique (combinaison de chiffres et de lettres) ;
- `IDREF` : une référence vers un ID ;
- `IDREFS` : une liste de références vers des ID (séparation par un blanc) ;
- `NMTOKEN` : un mot (donc pas de blanc) ;
- `NMTOKENS` : une liste de mots (séparation par un blanc) ;
- Une énumération de valeurs : chaque valeur est séparée par le caractère |.

Exemple


```
<!ELEMENT bibliography (book)+>❶
<!ELEMENT book (title, author, year, publisher, isbn, url?)>❷
<!ATTLIST book key   NMTOKEN  #REQUIRED>❸
<!ATTLIST book lang (fr | en) #REQUIRED>❹
<!ELEMENT title     (#PCDATA)>❺
<!ELEMENT author    (#PCDATA)>
<!ELEMENT year      (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT isbn      (#PCDATA)>
<!ELEMENT url       (#PCDATA)>
```

❶ Déclaration de l'élément `bibliography` devant contenir une suite non vide d'éléments `book`.

❷ Déclaration de l'élément `book` devant contenir les éléments `title`, `author`, …, `isbn` et `url`.

❸ ❹ Déclarations des attributs obligatoires `key` et `lang` de l'élément `book`.

❺ Déclaration de l'élément `title` devant contenir uniquement du texte.

Déclaration d'attributs identificateurs

Type ID

- Permet d'associer à un élément un identificateur unique
- Exemple :
  - Chaque produit dans un magasin doit avoir un code unique
- Déclaration :

  <!ATTLIST elt attr ID>

  Soit pour le magasin :

  <!ATTLIST produit code ID>
- Cette valeur doit être évidemment unique. Dans le cas contraire, le processeur XML renverra une erreur d'analyse lorsqu'il rencontrera un second identificateur identique

Exemple

- annuaire.dtd

```
<!ELEMENT annuaire (personne*)>
<!ELEMENT personne
(nom,prenom,email+)>
<!ATTLIST personne type (étudiant |
professeur | chanteur | musicien)
"étudiant">
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

Exemple

- annuaire.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE annuaire SYSTEM "annuaire.dtd">
<annuaire>
  <personne type="étudiant">
    <nom>HEUTE</nom>
    <prenom>Thomas</prenom>
    <email>webmaster@xmlfacile.com</email>
  </personne>
  <personne type="chanteur">
    <nom>CANTAT</nom>
    <prenom>Bertrand</prenom>
    <email>noir@desir.fr</email>
  </personne>
</annuaire>
```

- annuaire.dtd

```
<!ELEMENT annuaire (personne*)>
<!ELEMENT personne (nom,prenom,email+)>
<!ATTLIST personne type (étudiant | professeur | chanteur | musicien) "étudiant">
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

Inconvénients et avantages des DTD

ancienne car issue de l'univers SGML (*Standard Generalized Markup Language*). Elle a l'avantage d'être rapide à écrire, tout en présentant l'inconvénient d'être pauvre en possibilités de contrôle (typage de données, par exemple). Autre point négatif, les espaces de noms sont difficilement gérables car il faut intégrer, dans la grammaire, les préfixes, ce qui est contraire à l'esprit des espaces de noms.

Conclusion

- Une DTD définit des éléments et comment les **utiliser ensemble**,
- Les espaces de noms ne permettent d'établir qu'un **lien** entre un élément et un URI, mais **sans indiquer les contraintes d'utilisation.**