Introduction au langage XML (eXtensible Markup Language)

Belabed Amine

1ere Master SIC

2016-2017

Sommaire

Introduction

- XML comme HTML sont tous les deux conçus à partir de la norme SGML (Standard Generalized Markup Language).
- Mise au point par le XML Working Group sous la responsabilité du World Wide Web Consortium (W3C) dès 1996.
- Une recommandations Depuis le 10 février 1998, par le W3C avec les spécifications XML 1.0.

Introduction

- XML comme HTML sont tous les deux conçus à partir de la norme SGML (Standard Generalized Markup Language).
- Mise au point par le XML Working Group sous la responsabilité du World Wide Web Consortium (W3C) dès 1996.
- Une recommandations Depuis le 10 février 1998, par le W3C avec les spécifications XML 1.0.

Introduction

- XML comme HTML sont tous les deux conçus à partir de la norme SGML (Standard Generalized Markup Language).
- Mise au point par le XML Working Group sous la responsabilité du World Wide Web Consortium (W3C) dès 1996.
- Une recommandations Depuis le 10 février 1998, par le W3C avec les spécifications XML 1.0.

Exemple

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE annuaire SYSTEM "annuaire.dtd">
<annuaire type="pages blanches">
<entree>
<nom>Hadjila Fethallah </nom>
<telephone>06 03 02 01 00</telephone>
</entree>
<entree>
<nom>Midouni Djalal</nom>
<telephone>06 00 01 02 03</telephone>
</entree>
</annuaire>
```

L'entête XML

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

- **version** : version du XML utilisée la dernière version du langage, 1.1, date de février 2004;
- **encoding** : le jeu de codage de caractères utilisé, Par défaut, l'attribut encoding a la valeur UTF-8.
- **standalone**: dépendance du document par rapport à une déclaration de type de document.Si standalone est "yes", le processeur de l'application n'attend aucune déclaration de type de document extérieure au document.

L'entête XML

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

- **version** : version du XML utilisée la dernière version du langage, 1.1, date de février 2004;
- **encoding** : le jeu de codage de caractères utilisé, Par défaut, l'attribut encoding a la valeur UTF-8.
- **standalone**: dépendance du document par rapport à une déclaration de type de document.Si standalone est "yes", le processeur de l'application n'attend aucune déclaration de type de document extérieure au document.

L'entête XML

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

- **version** : version du XML utilisée la dernière version du langage, 1.1, date de février 2004;
- **encoding** : le jeu de codage de caractères utilisé, Par défaut, l'attribut encoding a la valeur UTF-8.
- **standalone**: dépendance du document par rapport à une déclaration de type de document.Si standalone est "yes", le processeur de l'application n'attend aucune déclaration de type de document extérieure au document.

Déclaration de type de document (DTD)

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

- Un autre type de document permettant de définir la structure d'un fichier XML est: *l'XML-Schema*, il s'utilise autrement.

Les éléments

- Un document XML peut se représenter sous la forme d'une arborescence d'éléments.
- Dans l'exemple précédent: `<annuaire>`, `<entree>`, `<nom>` sont des éléments.
- Une arborescence comporte une racine (unique): `<annuaire>` dans notre exemple.
- Les élément sont sensibles à la casse: `<entree>` est différent de `<Entree>`
- Un **élément vide** est un élément dont le contenu ("valeur de l'élément") est vide il s'écrit: `<elementvide></elementvide>` ou `<elementvide/>`.

Les éléments

- Un document XML peut se représenter sous la forme d'une arborescence d'éléments.
- Dans l'exemple précédent: `<annuaire>`, `<entree>`, `<nom>` sont des éléments.
- Une arborescence comporte une racine (unique): `<annuaire>` dans notre exemple.
- Les élément sont sensibles à la casse: `<entree>` est différent de `<Entree>`
- Un **élément vide** est un élément dont le contenu ("valeur de l'élément") est vide il s'écrit: `<elementvide></elementvide>` ou `<elementvide/>`.

Les éléments

- Un document XML peut se représenter sous la forme d'une arborescence d'éléments.
- Dans l'exemple précédent: `<annuaire>`, `<entree>`, `<nom>` sont des éléments.
- Une arborescence comporte une racine (unique): `<annuaire>` dans notre exemple.
- Les élément sont sensibles à la casse: `<entree>` est différent de `<Entree>`
- Un **élément vide** est un élément dont le contenu ("valeur de l'élément") est vide il s'écrit: `<elementvide></elementvide>` ou `<elementvide/>`.

Les éléments

- Un document XML peut se représenter sous la forme d'une arborescence d'éléments.
- Dans l'exemple précédent: `<annuaire>`, `<entree>`, `<nom>` sont des éléments.
- Une arborescence comporte une racine (unique): `<annuaire>` dans notre exemple.
- Les élément sont sensibles à la casse: `<entree>` est différent de `<Entree>`
- Un **élément vide** est un élément dont le contenu ("valeur de l'élément") est vide il s'écrit: `<elementvide></elementvide>` ou `<elementvide/>`.

Les éléments

- Un document XML peut se représenter sous la forme d'une arborescence d'éléments.
- Dans l'exemple précédent: `<annuaire>`, `<entree>`, `<nom>` sont des éléments.
- Une arborescence comporte une racine (unique): `<annuaire>` dans notre exemple.
- Les élément sont sensibles à la casse: `<entree>` est différent de `<Entree>`
- Un **élément vide** est un élément dont le contenu ("valeur de l'élément") est vide il s'écrit: `<elementvide></elementvide>` ou `<elementvide/>`.

Les attributs

- Tous les éléments peuvent contenir un ou plusieurs attributs.
- Dans l'exemple précédent: `<annuaire type="pages blanches">`, "type" est un attribut.
- Chaque élément ne peut contenir qu'une fois le même attribut.
- Un attribut ne peut être présent que dans la balise ouvrante de l'élément, l'écriture `</livre lang="en">` est fausse.

Les attributs

- Tous les éléments peuvent contenir un ou plusieurs attributs.
- Dans l'exemple précédent: `<annuaire type="pages blanches">`, "type" est un attribut.
- Chaque élément ne peut contenir qu'une fois le même attribut.
- Un attribut ne peut être présent que dans la balise ouvrante de l'élément, l'écriture `</livre lang="en">` est fausse.

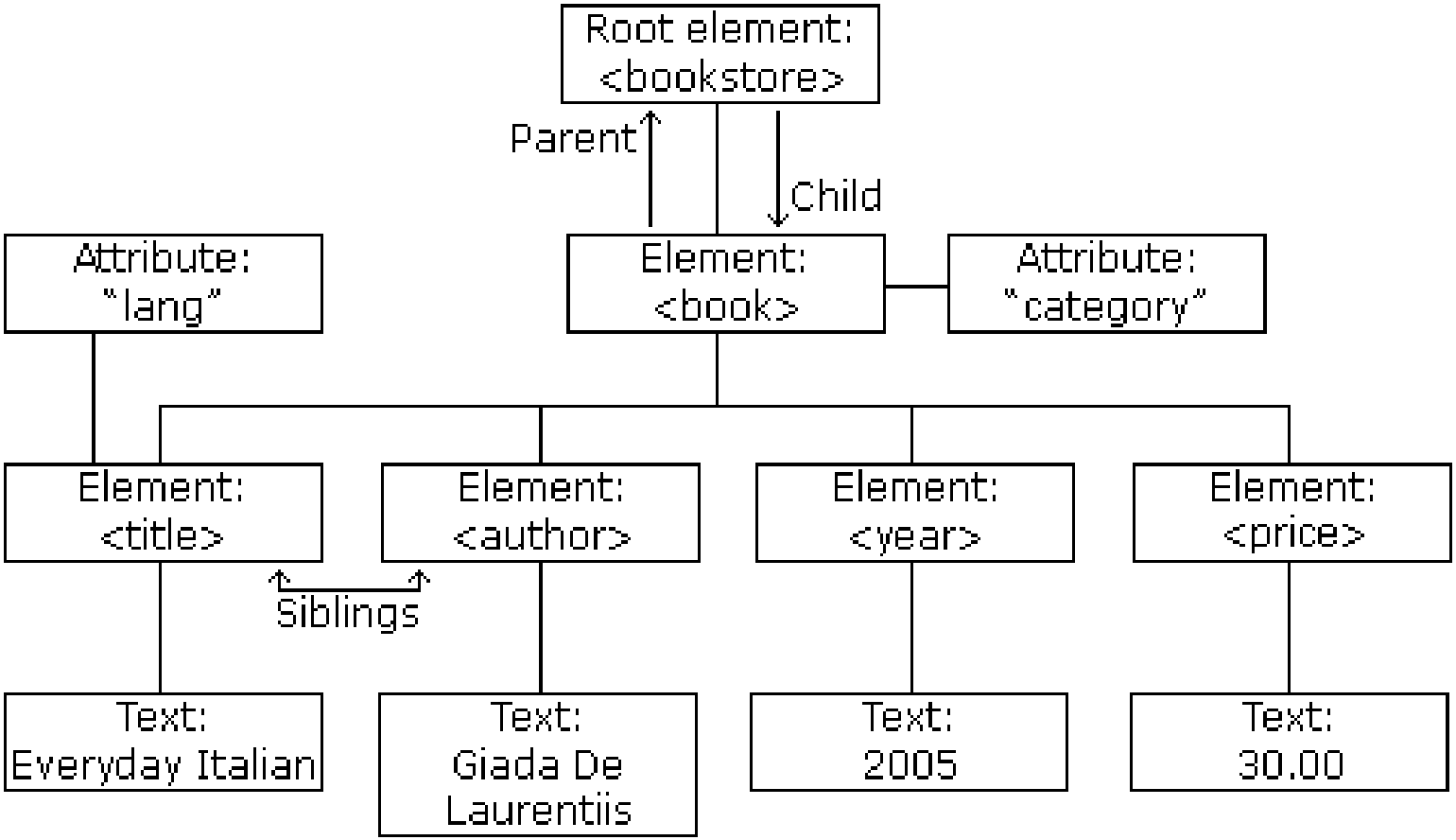
Les attributs

- Tous les éléments peuvent contenir un ou plusieurs attributs.
- Dans l'exemple précédent: `<annuaire type="pages blanches">`, "type" est un attribut.
- Chaque élément ne peut contenir qu'une fois le même attribut.
- Un attribut ne peut être présent que dans la balise ouvrante de l'élément, l'écriture `</livre lang="en">` est fausse.

Les attributs

- Tous les éléments peuvent contenir un ou plusieurs attributs.
- Dans l'exemple précédent: `<annuaire type="pages blanches">`, "type" est un attribut.
- Chaque élément ne peut contenir qu'une fois le même attribut.
- Un attribut ne peut être présent que dans la balise ouvrante de l'élément, l'écriture `</livre lang="en">` est fausse.

Exemple


Root element:
<bookstore>
Parent
Child
Attribute:
"lang"
Element:
<book>
Attribute:
"category"
Element:
<title>
Element:
<author>
Element:
<year>
Element:
<price>
Siblings
Text:
Everyday Italian
Text:
Giada De
Laurentiis
Text:
2005
Text:
30.00

Les espaces de noms

- Un document qui contient les informations générales d'une entreprise

```
<entreprise>
<nom> Entreprise inc </nom>
<adresse> une adresse </adresse>
</entreprise>
```

- Un document qui contient les informations sur le personnel de l'entreprise :

```
<personne>
<nom> Hadjila Fethallah </nom>
<fonction> PDG </fonction>
<telephone> 0102030405 </telephone>
</personne>
```

Les espaces de noms

- Un document qui contient les informations générales d'une entreprise

```
<entreprise>
<nom> Entreprise inc </nom>
<adresse> une adresse </adresse>
</entreprise>
```

- Un document qui contient les informations sur le personnel de l'entreprise :

```
<personne>
<nom> Hadjila Fethallah </nom>
<fonction> PDG </fonction>
<telephone> 0102030405 </telephone>
</personne>
```

Les espaces de noms

- la balise <nom> peut contenir deux types d'informations : nom de l'entreprise ou nom d'une personne.
- Si on désire créer un document unique décrivant l'entreprise et ses employés $\Longrightarrow$ un conflit au niveau de la balise <nom>.
- un moyen permettant d'identifier les balises est nécessaire: par exemple <personne:nom> et <entreprise:nom>
- C'est l 'objectif des espaces de nommage, ou XML Namespace (xmlns). Le principe est d'associer une URI (Uniform Resource Identifier) à un nom.

Les espaces de noms

- la balise <nom> peut contenir deux types d'informations : nom de l'entreprise ou nom d'une personne.
- Si on désire créer un document unique décrivant l'entreprise et ses employés $\Longrightarrow$ un conflit au niveau de la balise <nom>.
- un moyen permettant d'identifier les balises est nécessaire: par exemple <personne:nom> et <entreprise:nom>
- C'est l 'objectif des espaces de nommage, ou XML Namespace (xmlns). Le principe est d'associer une URI (Uniform Resource Identifier) à un nom.

Les espaces de noms

- la balise `<nom>` peut contenir deux types d'informations : nom de l'entreprise ou nom d'une personne.
- Si on désire créer un document unique décrivant l'entreprise et ses employés $\Longrightarrow$ un conflit au niveau de la balise `<nom>`.
- un moyen permettant d'identifier les balises est nécessaire: par exemple `<personne:nom>` et `<entreprise:nom>`
- C'est l 'objectif des espaces de nommage, ou XML Namespace (xmlns). Le principe est d'associer une URI (Uniform Resource Identifier) à un nom.

Les espaces de noms

- la balise `<nom>` peut contenir deux types d'informations : nom de l'entreprise ou nom d'une personne.
- Si on désire créer un document unique décrivant l'entreprise et ses employés $\Longrightarrow$ un conflit au niveau de la balise `<nom>`.
- un moyen permettant d'identifier les balises est nécessaire: par exemple `<personne:nom>` et `<entreprise:nom>`
- C'est l 'objectif des espaces de nommage, ou XML Namespace (xmlns). Le principe est d'associer une URI (Uniform Resource Identifier) à un nom.

Les espaces de noms

- Le document XML unique décrivant l'organisation de l'entreprise s'écrit alors :

```
<organisation
xmlns:entreprise="http://www.entreprise.org"
xmlns:personne ="http://www.personne.org">
<entreprise:nom>Entreprise inc</entreprise:nom>
<entreprise:adresse> une adresse </entreprise:adresse>
<personne:nom> Hadjila Fethallah</personne:nom>
<personne:fonction>PDG</personne:fonction>
</organisation>
```

Document bien formé et Document valide

- Un document XML bien formé est un document qui respecte la syntaxe XML.
- Un document XML valide est un document:
  - bien formé.
  - se conforme à la définition de la grammaire **(DTD ou Schéma)** à laquelle il est associé.

Document bien formé et Document valide

- Un document XML bien formé est un document qui respecte la syntaxe XML.
- Un document XML valide est un document:
  - bien formé.
  - se conforme à la définition de la grammaire **(DTD ou Schéma)** à laquelle il est associé.

Document bien formé et Document valide

- Un document XML bien formé est un document qui respecte la syntaxe XML.
- Un document XML valide est un document:
  - bien formé.
  - se conforme à la définition de la grammaire **(DTD ou Schéma)** à laquelle il est associé.

Document bien formé et Document valide

- Un document XML bien formé est un document qui respecte la syntaxe XML.
- Un document XML valide est un document:
  - bien formé.
  - se conforme à la définition de la grammaire **(DTD ou Schéma)** à laquelle il est associé.

La DTD interne

- C'est un DTD est incluse dans le document, son contenu est encadré par des caractères crochets '**[**' et '**]**'.
- Sa déclaration est sous forme:

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

- Exemple:

```
<?xml version="1.0"?>
<!DOCTYPE annuaire  [
<!ELEMENT annuaire (entree)>
<!ELEMENT entree (nom,telephone*)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT telephone (#PCDATA)>
] >
<entree>
...
</entree>
```

La DTD interne

- C'est un DTD est incluse dans le document, son contenu est encadré par des caractères crochets '**[**' et '**]**'.
- Sa déclaration est sous forme:

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

- Exemple:

```
<?xml version="1.0"?>
<!DOCTYPE annuaire  [
<!ELEMENT annuaire (entree)>
<!ELEMENT entree (nom,telephone*)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT telephone (#PCDATA)>
] >
<entree>
...
</entree>
```

La DTD interne

- C'est un DTD est incluse dans le document, son contenu est encadré par des caractères crochets '**[**' et '**]**'.
- Sa déclaration est sous forme:

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

- Exemple:

```
<?xml version="1.0"?>
<!DOCTYPE annuaire  [
<!ELEMENT annuaire (entree)>
<!ELEMENT entree (nom,telephone*)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT telephone (#PCDATA)>
] >
<entree>
...
</entree>
```

La DTD externe

- La déclaration DTD est faite dans un fichier à part.
- Sa déclaration est sous forme:

```
<!DOCTYPE root-element SYSTEM "file.dtd" >
ou:
<!DOCTYPE root-element PUBLIC "URL" >
```

- Exemple:

```
<?xml version="1.0"?>
<!DOCTYPE entree SYSTEM "entree.dtd" >
<entree>
...
</entree>
```

La DTD externe

- La déclaration DTD est faite dans un fichier à part.
- Sa déclaration est sous forme:

```
<!DOCTYPE root-element SYSTEM "file.dtd" >
ou:
<!DOCTYPE root-element PUBLIC "URL" >
```

- Exemple:

```
<?xml version="1.0"?>
<!DOCTYPE entree SYSTEM "entree.dtd" >
<entree>
...
</entree>
```

La DTD externe

- La déclaration DTD est faite dans un fichier à part.
- Sa déclaration est sous forme:

```
<!DOCTYPE root-element SYSTEM "file.dtd" >
ou:
<!DOCTYPE root-element PUBLIC "URL" >
```

- Exemple:

```
<?xml version="1.0"?>
<!DOCTYPE entree SYSTEM "entree.dtd" >
<entree>
...
</entree>
```

DTD: Types de données

- Les DTD définissent 10 types de données. Les deux types les plus fréquemment utilisés sont : **PCDATA** et **CDATA** :
  - **PCDATA** : signifie Parsed Character Data (ou chaîne de caractères parsée). C'est le texte contenu dans un élément. Dans l'élément suivant : `<element>texte</element>`, " texte " est du type PCDATA.
  - **CDATA** : signifie Character Data ou chaîne de caractères. des chaîne de caractères qui ne seront pas parsées lors de la validation. On les utilise dans les attributs : ex : `<element attribut="<hello/>">`, `<hello/>` est du type CDATA et sera donc traitée comme une chaîne de caractère, et non une balise XML.
  - **ID et IDREF** : permettent de lier différentes parties d'un document XML.

DTD: Types de données

- Les DTD définissent 10 types de données. Les deux types les plus fréquemment utilisés sont : **PCDATA** et **CDATA** :
  - **PCDATA** : signifie Parsed Character Data (ou chaîne de caractères parsée). C'est le texte contenu dans un élément. Dans l'élément suivant : `<element>texte</element>`, " texte " est du type PCDATA.
  - **CDATA** : signifie Character Data ou chaîne de caractères. des chaîne de caractères qui ne seront pas parsées lors de la validation. On les utilise dans les attributs : ex : `<element attribut="<hello/>">`, `<hello/>` est du type CDATA et sera donc traitée comme une chaîne de caractère, et non une balise XML.
  - **ID et IDREF** : permettent de lier différentes parties d'un document XML.

DTD: Types de données

- Les DTD définissent 10 types de données. Les deux types les plus fréquemment utilisés sont : **PCDATA** et **CDATA** :
  - **PCDATA** : signifie Parsed Character Data (ou chaîne de caractères parsée). C'est le texte contenu dans un élément. Dans l'élément suivant : `<element>texte</element>`, " texte " est du type PCDATA.
  - **CDATA** : signifie Character Data ou chaîne de caractères. des chaîne de caractères qui ne seront pas parsées lors de la validation. On les utilise dans les attributs : ex : `<element attribut="<hello/>">`, `<hello/>` est du type CDATA et sera donc traitée comme une chaîne de caractère, et non une balise XML.
  - **ID et IDREF** : permettent de lier différentes parties d'un document XML.

DTD: Types de données

- Les DTD définissent 10 types de données. Les deux types les plus fréquemment utilisés sont : **PCDATA** et **CDATA** :
  - **PCDATA** : signifie Parsed Character Data (ou chaîne de caractères parsée). C'est le texte contenu dans un élément. Dans l'élément suivant : `<element>texte</element>`, " texte " est du type PCDATA.
  - **CDATA** : signifie Character Data ou chaîne de caractères. des chaîne de caractères qui ne seront pas parsées lors de la validation. On les utilise dans les attributs : ex : `<element attribut="<hello/>">`, `<hello/>` est du type CDATA et sera donc traitée comme une chaîne de caractère, et non une balise XML.
  - **ID et IDREF** : permettent de lier différentes parties d'un document XML.

DTD: les éléments

- `<!ELEMENT nom_balise spécification_contenu>`

Type	DTD	XML
Elément vide	`<!ELEMENT elt EMPTY>`	`<elt/>`
Elément contenant du texte	`<!ELEMENT elt (#PCDATA)>`	`<elt>texte</elt>`

Elément avec sous éléments	`<!ELEMENT elt (sous-elt)>`
	`<!ELEMENT sous-elt EMPTY>`
		`<elt>`
		`  <sous-elt/>`
		`</elt>`

Elément avec plusieurs sous éléments	`<!ELEMENT elt (s1, s2)>`
	`<!ELEMENT s1 EMPTY>`
	`<!ELEMENT s2 EMPTY>`
		`<elt>`
		`  <s1/>`
		`  <s2/>`
		`</elt>`

Elément avec contenu variable	`<!ELEMENT elt (#PCDATA|s1)>`
	`<!ELEMENT s1 EMPTY>`
		`<elt>texte</elt>`
		ou
		`<elt><s1/></elt>`

Elément à contenu non défini	`<!ELEMENT elt ANY>`
	`<!ELEMENT s1 EMPTY>`
		`<elt>texte</elt>`
		ou
		`<elt><s1/></elt>`
		etc.

DTD: les cardinalités des éléments

- le sous élément s1 peut avoir une et une seule

  ```
  <!ELEMENT element (s1)>
  ```

- le sous élément s1 peut avoir 0 à n occurrences (*).

  ```
  <!ELEMENT element (s1*)>
  ```

- le sous élément s1 peut avoir 0 ou 1 occurrence (?).

  ```
  <!ELEMENT element (s1?)>
  ```

- le sous élément s1 doit avoir au moins 1 occurrence.

  ```
  <!ELEMENT element (s1+)>
  ```

DTD: les cardinalités des éléments

- le sous élément s1 peut avoir une et une seule

  `<!ELEMENT element (s1)>`
- le sous élément s1 peut avoir 0 à n occurrences (*).

  `<!ELEMENT element (s1*)>`
- le sous élément s1 peut avoir 0 ou 1 occurrence (?).

  `<!ELEMENT element (s1?)>`
- le sous élément s1 doit avoir au moins 1 occurrence.

  `<!ELEMENT element (s1+)>`

DTD: les cardinalités des éléments

- le sous élément s1 peut avoir une et une seule

  ```
  <!ELEMENT element (s1)>
  ```

- le sous élément s1 peut avoir 0 à n occurrences (*).

  ```
  <!ELEMENT element (s1*)>
  ```

- le sous élément s1 peut avoir 0 ou 1 occurrence (?).

  ```
  <!ELEMENT element (s1?)>
  ```

- le sous élément s1 doit avoir au moins 1 occurrence.

  ```
  <!ELEMENT element (s1+)>
  ```

DTD: les cardinalités des éléments

- le sous élément s1 peut avoir une et une seule

  ```
  <!ELEMENT element (s1)>
  ```

- le sous élément s1 peut avoir 0 à n occurrences (*).

  ```
  <!ELEMENT element (s1*)>
  ```

- le sous élément s1 peut avoir 0 ou 1 occurrence (?).

  ```
  <!ELEMENT element (s1?)>
  ```

- le sous élément s1 doit avoir au moins 1 occurrence.

  ```
  <!ELEMENT element (s1+)>
  ```

DTD: les attributs

- Déclaration:

  `<!ATTLIST nom-element nom-attrib1 type default-value nom-element nom-attrib2 type default-value ...>`

- Exemple: `<!ATTLIST Livre lang CDATA>`

DTD	XML
<!ELEMENT elt EMPTY> <!ATTLIST elt attribut CDATA "defaut">	<elt attribut="valeur"/> Par défaut : <elt attribut="defaut"/>
<!ELEMENT elt EMPTY> <!ATTLIST elt attribut ("v1"|"v2") "v1"> 2 valeurs possibles : "v1" et "v2"	<elt attribut="v2"/> ou (défaut) : <elt attribut="v1"/>
<!ELEMENT elt EMPTY> <!ATTLIST elt att1 CDATA "d1" elt att2 CDATA "d2">	<elt att1="a" att2="b"/> par défaut: <elt att1="d1" att2="d2"/>

les Options d'attributs

- Quelques options:

Type	DTD	XML
Attribut obligatoire	<!ELEMENT elt EMPTY> <!ATTLIST elt att CDATA #REQUIRED>	<elt att="a"/>
Attribut non obligatoire	<!ELEMENT elt EMPTY> <!ATTLIST elt att CDATA #IMPLIED>	<elt att="a"/> <elt/>
Attribut à valeur fixe	<!ELEMENT elt EMPTY> <!ATTLIST elt att CDATA #FIXED "a">	<elt att="a"/>: OK <elt att="b"/>: non

les Entités

- Les entités peuvent être considérées comme des variables :
  `<!ENTITY nom "valeur">`
  `<!ENTITY nom SYSTEM "file-ou-URL">`
- Exemple:
  - `<!ENTITY eacute "&#233;">` dans XML s'écrit:
    `<balise> &eacute;gale</balise>` (égale)
  - `<!ENTITY explication SYSTEM "file.xml">` dans XML l'écriture :
    `<citation> &explication; </citation>` est remplacée par:
    `<citation> contenu du fichier file.xml </citation>`

les Entités

- Les entités peuvent être considérées comme des variables :
  `<!ENTITY nom "valeur">`
  `<!ENTITY nom SYSTEM "file-ou-URL">`
- Exemple:
  - `<!ENTITY eacute "&#233;">` dans XML s'écrit:
    `<balise> &eacute;gale</balise>` (égale)
  - `<!ENTITY explication SYSTEM "file.xml">` dans XML l'écriture :
    `<citation> &explication; </citation>` est remplacée par:
    `<citation> contenu du fichier file.xml </citation>`

les Entités

- Les entités peuvent être considérées comme des variables :
  `<!ENTITY nom "valeur">`
  `<!ENTITY nom SYSTEM "file-ou-URL">`
- Exemple:
  - `<!ENTITY eacute "&#233;">` dans XML s'écrit:
    `<balise> &eacute;gale</balise>` (égale)
  - `<!ENTITY explication SYSTEM "file.xml">` dans XML l'écriture :
    `<citation> &explication; </citation>` est remplacée par:
    `<citation> contenu du fichier file.xml </citation>`

les Entités

- Les entités peuvent être considérées comme des variables :
  `<!ENTITY nom "valeur">`
  `<!ENTITY nom SYSTEM "file-ou-URL">`
- Exemple:
  - `<!ENTITY eacute "&#233;">` dans XML s'écrit:
    `<balise> &eacute;gale</balise>` (égale)
  - `<!ENTITY explication SYSTEM "file.xml">` dans XML l'écriture :
    `<citation> &explication; </citation>` est remplacée par:
    `<citation> contenu du fichier file.xml </citation>`

Les identifiants

- Un attribut de type ID permet d'attribuer un identifiant à un élément d'un document XML.
- Cet élément peut être ensuite référencé dans une autre partie du même document en utilisant un attribut de type IDREF.

Les identifiants

- Exemple:

```
 <!ELEMENT personne (#PCDATA)>
<!ELEMENT parent (fils*)>
<!ELEMENT fils EMPTY>
<!ATTLIST personne identifiant ID #REQUIRED>
<!ATTLIST parent identifiant IDREF #REQUIRED>
<!ATTLIST fils identifiant IDREF #REQUIRED>
<!-- XML -->
<personne identifiant="A">Mohamed</personne>
<personne identifiant="B">Amine</personne>
<parent identifiant="A">
<fils identifiant="B"/>
</parent>
```

XML-Schema

- XML schema est une amélioration des DTD.
- il permet la spécification :
  - Les types d'éléments.
  - Les relations de structuration.
  - Les relations de spécialisation.
  - Le typage des données (string, entier, réel..).
  - Les restrictions (cardinalités), et les collections d'éléments.
- Un schéma XML se compose essentiellement de *déclarations d'éléments* et d'*attributs* et de *définitions de types*.

XML-Schema

- XML schema est une amélioration des DTD.
- il permet la spécification :
  - Les types d'éléments.
  - Les relations de structuration.
  - Les relations de spécialisation.
  - Le typage des données (string, entier, réel..).
  - Les restrictions (cardinalités), et les collections d'éléments.
- Un schéma XML se compose essentiellement de *déclarations d'éléments* et d'*attributs* et de *définitions de types*.

XML-Schema

- XML schema est une amélioration des DTD.
- il permet la spécification :
  - Les types d'éléments.
  - Les relations de structuration.
  - Les relations de spécialisation.
  - Le typage des données (string, entier, réel..).
  - Les restrictions (cardinalités), et les collections d'éléments.
- Un schéma XML se compose essentiellement de *déclarations d'éléments* et d'*attributs* et de *définitions de types*.

XML-Schema

- XML schema est une amélioration des DTD.
- il permet la spécification :
  - Les types d'éléments.
  - Les relations de structuration.
  - Les relations de spécialisation.
  - Le typage des données (string, entier, réel..).
  - Les restrictions (cardinalités), et les collections d'éléments.
- Un schéma XML se compose essentiellement de *déclarations d'éléments* et d'*attributs* et de *définitions de types*.

XML-Schema

- XML schema est une amélioration des DTD.
- il permet la spécification :
  - Les types d'éléments.
  - Les relations de structuration.
  - Les relations de spécialisation.
  - Le typage des données (string, entier, réel..).
  - Les restrictions (cardinalités), et les collections d'éléments.
- Un schéma XML se compose essentiellement de *déclarations d'éléments* et d'*attributs* et de *définitions de types*.

XML-Schema

- XML schema est une amélioration des DTD.
- il permet la spécification :
  - Les types d'éléments.
  - Les relations de structuration.
  - Les relations de spécialisation.
  - Le typage des données (string, entier, réel..).
  - Les restrictions (cardinalités), et les collections d'éléments.
- Un schéma XML se compose essentiellement de *déclarations d'éléments* et d'*attributs* et de *définitions de types*.

XML-Schema

- XML schema est une amélioration des DTD.
- il permet la spécification :
  - Les types d'éléments.
  - Les relations de structuration.
  - Les relations de spécialisation.
  - Le typage des données (string, entier, réel..).
  - Les restrictions (cardinalités), et les collections d'éléments.
- Un schéma XML se compose essentiellement de *déclarations d'éléments* et d'*attributs* et de *définitions de types*.

XML-Schema

- XML schema est une amélioration des DTD.
- il permet la spécification :
  - Les types d'éléments.
  - Les relations de structuration.
  - Les relations de spécialisation.
  - Le typage des données (string, entier, réel..).
  - Les restrictions (cardinalités), et les collections d'éléments.
- Un schéma XML se compose essentiellement de *déclarations d'éléments* et d'*attributs* et de *définitions de types*.

Référer à un XML-Schema dans un document XML

- Le schèma pour ce document est :"schema.xsd"

```
<elem-racine
xmlns="URI/namespace"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:schemaLocation ="URI/schema.xsd">
```

Si notre document ne définit pas un espace de nom :

```
<elem-racine
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:noNamespaceSchemaLocation ="URI/schema.xsd">
```

La Structure d'un XML-Schema

- Un schéma XML est un fichier XML dont l'élément racine est : `<xsd:schema>`
- La structure d'un schéma est comme suit:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!-- Déclarations d'éléments, d'attributs et types -->
  ...
</xsd:schema>
```

Eléments et Types de données

- La déclaration la plus simple d'un élément prend la forme suivante: `<xsd:element name="element" type="type"/>`
- Exemple: `<xsd:element name="title" type="xsd:string"/>`
- XML-Schema définit plus de 40 types de données: string, integer, date, year, CDATA, float, double, binary, ENTITIES, token, byte, etc.
- il est possible de créer un type de données totalement nouveau, de restreindre ou d'étendre un type de données existant

Eléments et Types de données

- La déclaration la plus simple d'un élément prend la forme suivante: `<xsd:element name="element" type="type"/>`
- Exemple: `<xsd:element name="title" type="xsd:string"/>`
- XML-Schema définit plus de 40 types de données: string, integer, date, year, CDATA, float, double, binary, ENTITIES, token, byte, etc.
- il est possible de créer un type de données totalement nouveau, de restreindre ou d'étendre un type de données existant

Eléments et Types de données

- La déclaration la plus simple d'un élément prend la forme suivante: `<xsd:element name="element" type="type"/>`
- Exemple: `<xsd:element name="title" type="xsd:string"/>`
- XML-Schema définit plus de 40 types de données: string, integer, date, year, CDATA, float, double, binary, ENTITIES, token, byte, etc.
- il est possible de créer un type de données totalement nouveau, de restreindre ou d'étendre un type de données existant

Eléments et Types de données

- La déclaration la plus simple d'un élément prend la forme suivante: `<xsd:element name="element" type="type"/>`
- Exemple: `<xsd:element name="title" type="xsd:string"/>`
- XML-Schema définit plus de 40 types de données: string, integer, date, year, CDATA, float, double, binary, ENTITIES, token, byte, etc.
- il est possible de créer un type de données totalement nouveau, de restreindre ou d'étendre un type de données existant

Eléments et Types de données

- Lors de la déclaration d'un élément, il est possible de décrire explicitement le type:

```
<xsd:element name="element"> (Type simple)
  <xsd:simpleType>
  ...
  </xsd:simpleType>
</xsd:element>
```

ou:

```
<xsd:element name="element"> (Type complexe)
  <xsd:complexType>
  ...
  </xsd:complexType>
</xsd:element>
```

Les Type Simples

- Les types simples peuvent être utilisés pour *les éléments* ou les *attributs*.
- Les types simples sont introduits par l'élément `xsd:simpleType`.
- Un type simple est souvent obtenu par:
  - Restriction d'un autre type défini.
  - Union d'autres types simples.
  - L'opérateur de listes.

Les Type Simples

- Les types simples peuvent être utilisés pour *les éléments* ou les *attributs*.
- Les types simples sont introduits par l'élément `xsd:simpleType`.
- Un type simple est souvent obtenu par:
  - Restriction d'un autre type défini.
  - Union d'autres types simples.
  - L'opérateur de listes.

Les Type Simples

- Les types simples peuvent être utilisés pour *les éléments* ou les *attributs*.
- Les types simples sont introduits par l'élément `xsd:simpleType`.
- Un type simple est souvent obtenu par:
  - Restriction d'un autre type défini.
  - Union d'autres types simples.
  - L'opérateur de listes.

Les Type Simples

- Les types simples peuvent être utilisés pour *les éléments* ou les *attributs*.
- Les types simples sont introduits par l'élément `xsd:simpleType`.
- Un type simple est souvent obtenu par:
  - Restriction d'un autre type défini.
  - Union d'autres types simples.
  - L'opérateur de listes.

Les Type Simples

- Les types simples peuvent être utilisés pour *les éléments* ou les *attributs*.
- Les types simples sont introduits par l'élément `xsd:simpleType`.
- Un type simple est souvent obtenu par:
  - Restriction d'un autre type défini.
  - Union d'autres types simples.
  - L'opérateur de listes.

Les Type Simples

- Les types simples peuvent être utilisés pour *les éléments* ou les *attributs*.
- Les types simples sont introduits par l'élément `xsd:simpleType`.
- Un type simple est souvent obtenu par:
  - Restriction d'un autre type défini.
  - Union d'autres types simples.
  - L'opérateur de listes.

Les Type Simples: Exemples

- Exemple 1 : Restriction de type.

```
<xsd:simpleType name="choixOuiNon">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="oui"/>
<xsd:enumeration value="non"/>
</xsd:restriction>
</xsd:simpleType>
```

- Utilisation:

```
<xsd:element name="choix" type="choixOuiNon"/>
```

Les Type Simples: Exemples

- Exemple 1 : Restriction de type.

```
<xsd:simpleType name="choixOuiNon">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="oui"/>
<xsd:enumeration value="non"/>
</xsd:restriction>
</xsd:simpleType>
```

- Utilisation:

```
<xsd:element name="choix" type="choixOuiNon"/>
```

Les Type Simples: Exemples

- Exemple 2 : Opérateur de liste.

```
<xsd:simpleType name="numéroDeTéléphone">
  <xsd:list itemType="xsd:unsignedByte" />
</xsd:simpleType>
```

- Utilisation: après la déclaration d'un elément téléphone de type "numéroDeTéléphone".

```
<téléphone>01 44 27 60 11</téléphone>
```

Les Type Simples: Exemples

- Exemple 2 : Opérateur de liste.

```
<xsd:simpleType name="numéroDeTéléphone">
  <xsd:list itemType="xsd:unsignedByte" />
</xsd:simpleType>
```

- Utilisation: après la déclaration d'un elément téléphone de type "numéroDeTéléphone".

```
<téléphone>01 44 27 60 11</téléphone>
```

Les Type Simples: Exemples

- Exemple 3 : Union de types.

```
<xsd:simpleType name="numDeTéléphoneMnémoTechnique">
<xsd:union memberTypes="xsd:string numéroDeTéléphone"/>
</xsd:simpleType>
```

- Utilisation:

```
<téléphone> 17 </téléphone>
<téléphone>Police</téléphone>
```

Les Type Simples: Exemples

- Exemple 3 : Union de types.

```
<xsd:simpleType name="numDeTéléphoneMnémoTechnique">
<xsd:union memberTypes="xsd:string numéroDeTéléphone"/>
</xsd:simpleType>
```

- Utilisation:

```
<téléphone> 17 </téléphone>
<téléphone>Police</téléphone>
```

Les Type Comlexes

- Un élément de type complexe, est un élément qui contient des sous éléments.
- Les types complexes sont introduits par l'élément `xsd:complexType`.
- Un type complexe est souvent obtenu par:
  - Opérateur de séquence.
  - Opérateur de choix.

Les Type Comlexes

- Un élément de type complexe, est un élément qui contient des sous éléments.
- Les types complexes sont introduits par l'élément `xsd:complexType`.
- Un type complexe est souvent obtenu par:
  - Opérateur de séquence.
  - Opérateur de choix.

Les Type Comlexes

- Un élément de type complexe, est un élément qui contient des sous éléments.
- Les types complexes sont introduits par l'élément `xsd:complexType`.
- Un type complexe est souvent obtenu par:
  - Opérateur de séquence.
  - Opérateur de choix.

Les Type Comlexes

- Un élément de type complexe, est un élément qui contient des sous éléments.
- Les types complexes sont introduits par l'élément `xsd:complexType`.
- Un type complexe est souvent obtenu par:
  - Opérateur de séquence.
  - Opérateur de choix.

Les Type Comlexes

- Un élément de type complexe, est un élément qui contient des sous éléments.
- Les types complexes sont introduits par l'élément `xsd:complexType`.
- Un type complexe est souvent obtenu par:
  - Opérateur de séquence.
  - Opérateur de choix.

Les Type Comlexes: Exemples

- Exemple 1: Opérateur de séquence.

```
<entree>
<nom>string</nom>
<telephone> decimal</telephone>
</entree>
```

Le type de l'élémént entree se déclare comme suit:

```
<xsd:complexType name="typeEntree">
<xsd:sequence>
<xsd:element name="nom" type="xsd:string"/>
<xsd:element name="telephone" type="xsd:decimal"/>
</xsd:sequence>
</xsd:complexType>
```

Les Type Comlexes

- Exemple 2: Opérateur de choix.
  Le type d'un élément "publication" qui contient au choix les élément: book,article ou report.

```
<xsd:element name="publication">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element ref="book"/>
      <xsd:element ref="article"/>
      <xsd:element ref="report"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

les Cardinalités

- L'élément `<xsd:element>` possède deux attributs optionnels:
  - `minoccurs`:Nombre minimal d'occurrences d'un élément.
  - `maxoccurs`:Nombre maximal d'occurrences d'un élément
- Exemple:

```
<xsd:element name="element">
<xsd:complexType>
    <xsd:sequence>
      <xsd:element name="s1"
                   minoccurs="0"
                   maxoccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

les Cardinalités

- L'élément `<xsd:element>` possède deux attributs optionnels:
  - `minoccurs`:Nombre minimal d'occurrences d'un élément.
  - `maxoccurs`:Nombre maximal d'occurrences d'un élément
- Exemple:

```
<xsd:element name="element">
<xsd:complexType>
    <xsd:sequence>
      <xsd:element name="s1"
                   minoccurs="0"
                   maxoccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

les Cardinalités

- L'élément `<xsd:element>` possède deux attributs optionnels:
  - `minoccurs`:Nombre minimal d'occurrences d'un élément.
  - `maxoccurs`:Nombre maximal d'occurrences d'un élément
- Exemple:

```
<xsd:element name="element">
<xsd:complexType>
    <xsd:sequence>
      <xsd:element name="s1"
                   minoccurs="0"
                   maxoccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

les Cardinalités

- L'élément <xsd:element> possède deux attributs optionnels:
  - minoccurs:Nombre minimal d'occurrences d'un élément.
  - maxoccurs:Nombre maximal d'occurrences d'un élément
- Exemple:

```
<xsd:element name="element">
<xsd:complexType>
    <xsd:sequence>
      <xsd:element name="s1"
                   minoccurs="0"
                   maxoccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

Les attributs

- La définition d'attributs associés à un élément se fait dans un élément `<xsd:attribute>`.
- Chaque élément ¡xsd:attribute¿ possède les attributs suivants :
  - **Name** : le nom de l'attribut.
  - **Type** : le type de l'attribut. Par exemple xsd:string, xsd:boolean,...
  - **Use** : permet de préciser si l'attribut est obligatoire ou optionnel. les Valeurs possibles "`required`" (obligatoire), "`optional`" (optionnel).
  - **Default (fixed)** : valeur par défaut de l'attribut.

Les attributs

- La définition d'attributs associés à un élément se fait dans un élément `<xsd:attribute>`.
- Chaque élément ¡xsd:attribute¿ possède les attributs suivants :
  - **Name** : le nom de l'attribut.
  - **Type** : le type de l'attribut. Par exemple xsd:string, xsd:boolean,...
  - **Use** : permet de préciser si l'attribut est obligatoire ou optionnel. les Valeurs possibles "`required` " (obligatoire), "`optional`" (optionnel).
  - **Default (fixed)** : valeur par défaut de l'attribut.

Les attributs

- La définition d'attributs associés à un élément se fait dans un élément `<xsd:attribute>`.
- Chaque élément ¡xsd:attribute¿ possède les attributs suivants :
  - **Name** : le nom de l'attribut.
  - **Type** : le type de l'attribut. Par exemple xsd:string, xsd:boolean,...
  - **Use** : permet de préciser si l'attribut est obligatoire ou optionnel. les Valeurs possibles "`required` " (obligatoire), "`optional`" (optionnel).
  - **Default (fixed)** : valeur par défaut de l'attribut.

Les attributs

- La définition d'attributs associés à un élément se fait dans un élément `<xsd:attribute>`.
- Chaque élément ¡xsd:attribute¿ possède les attributs suivants :
  - **Name** : le nom de l'attribut.
  - **Type** : le type de l'attribut. Par exemple xsd:string, xsd:boolean,...
  - **Use** : permet de préciser si l'attribut est obligatoire ou optionnel. les Valeurs possibles "`required` " (obligatoire), "`optional`" (optionnel).
  - **Default (fixed)** : valeur par défaut de l'attribut.

Les attributs

- La définition d'attributs associés à un élément se fait dans un élément `<xsd:attribute>`.
- Chaque élément ¡xsd:attribute¿ possède les attributs suivants :
  - **Name** : le nom de l'attribut.
  - **Type** : le type de l'attribut. Par exemple xsd:string, xsd:boolean,...
  - **Use** : permet de préciser si l'attribut est obligatoire ou optionnel. les Valeurs possibles "`required` " (obligatoire), "`optional`" (optionnel).
  - **Default (fixed)** : valeur par défaut de l'attribut.

Les attributs

- La définition d'attributs associés à un élément se fait dans un élément `<xsd:attribute>`.
- Chaque élément ¡xsd:attribute¿ possède les attributs suivants :
  - **Name** : le nom de l'attribut.
  - **Type** : le type de l'attribut. Par exemple xsd:string, xsd:boolean,...
  - **Use** : permet de préciser si l'attribut est obligatoire ou optionnel. les Valeurs possibles "`required` " (obligatoire), "`optional`" (optionnel).
  - **Default (fixed)** : valeur par défaut de l'attribut.

Les attributs : Exemple

- <element att="hello" at2="true"/>."att" est optionnel avec valeur par défaut "a". "at2" est obligatoire et valeur par défaut " true ".

```
<xsd:element name="element">
   <xsd:complexType>
        <xsd:attribute name="att"
                       type="xsd:string"
                       use="Optional"
                       default="a"/>
         <xsd:attribute name="at2"
                       type="xsd:boolean"
                       use="required"
                       default="true"/>
</xsd:complexType>
</xsd:element>
```

les Identifiants

- Comme pour les DTD, il est possible d'attribuer un identifiant aux éléments (attribut de type **ID**),pour les référencer dans la suite du document XML (attribut de type **IDREF**).
- Exemple de définition d'élément possédant un attribut identifiant :

```
<xsd:element name="element1">
<xsd:complexType>
<xsd:attribute name="id" type="ID" use="required"/>
</xsd:complexType>
</xsd:element>
```

les Identifiants

- Exemple de définition d'élément référençant un autre élément possédant un identifiant :

```
<xsd:element name="element2">
<xsd:complexType>
<xsd:attribute name="ref" type="IDREF" use="required"/>
</xsd:complexType>
</xsd:element>
```

- Exemple dans un document XML, où un élément <element2> fait référence à un élément <element1>:

```
<element1 id="A"/>
<element2 ref="A"/>
```

En savoir plus

- http://www.w3.org/XML/
- http://xmlfr.org/
- http://www.w3schools.com/xml/
- http://xml.developpez.com/
- http://www.yoyodesign.org/doc/w3c/xml11/