جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

Université Abderrahmane Mira de Béjaia
Faculté des Sciences Exactes
Département d'Informatique

Module : Services Web (M2 – GL – ReSyD – IA (S3))

Cours 02 : XML

Présenté par Dr BRAHAMI EL BOUHISSI H.

2017-2018

Structuration des données : quels besoins ?

Besoin d'échanger des données entre acteurs :

– Données bancaires
– Données de remboursement de santé
– Description d'une recherche d'itinéraire dans Google Maps
– La réponse associée : description d'un itinéraire
– ...

⇒ Nécessité d'un format de structuration universel :
– Permettant de représenter toutes sortes de données structurées (non limitant à un certain type de données)
– Lisible sur tous les systèmes
– Le plus simple que possible

Les deux formats d'échange du web

- **XML (1998)**

  – Généralisation de HTML pour représenter toutes sortes de données et pas seulement des pages web

  – Origine : W3C avec un soutien industriel fort (Microsoft)

- **JSON (2006)**

  – Sous-ensemble de JavaScript

XML : Définition

- XML est un **langage de structuration de** document.
- XML est **extensible: on peut créer autant de balises que l'on souhaite.**
  - Les balises HTML servent à formater les informations.
  - Les balises XML ne servent qu'à structurer les documents.
- Un document XML ne possède aucune information sur la manière dont il doit être affiché sur un écran d'ordinateur.

Exemple d'un document XML

demo.xml

```
1- <?xml version="1.0"?>
2- <demoXML>
3- <message>Voici du XML</message>
4- </demoXML>
```

Explication:

1- prologue: déclaration du type de document.
2- la balise ouvrante de l'élément racine du document
3- un sous élément avec un contenu
4- la balise fermante de l'élément racine du document

Exemple HTML / XML

Donald Knuth
Date de naissance : 10 janvier 1938
Adresse : Stanford University
CA 94305
Etats-Unis
Profession : informaticien

HTML

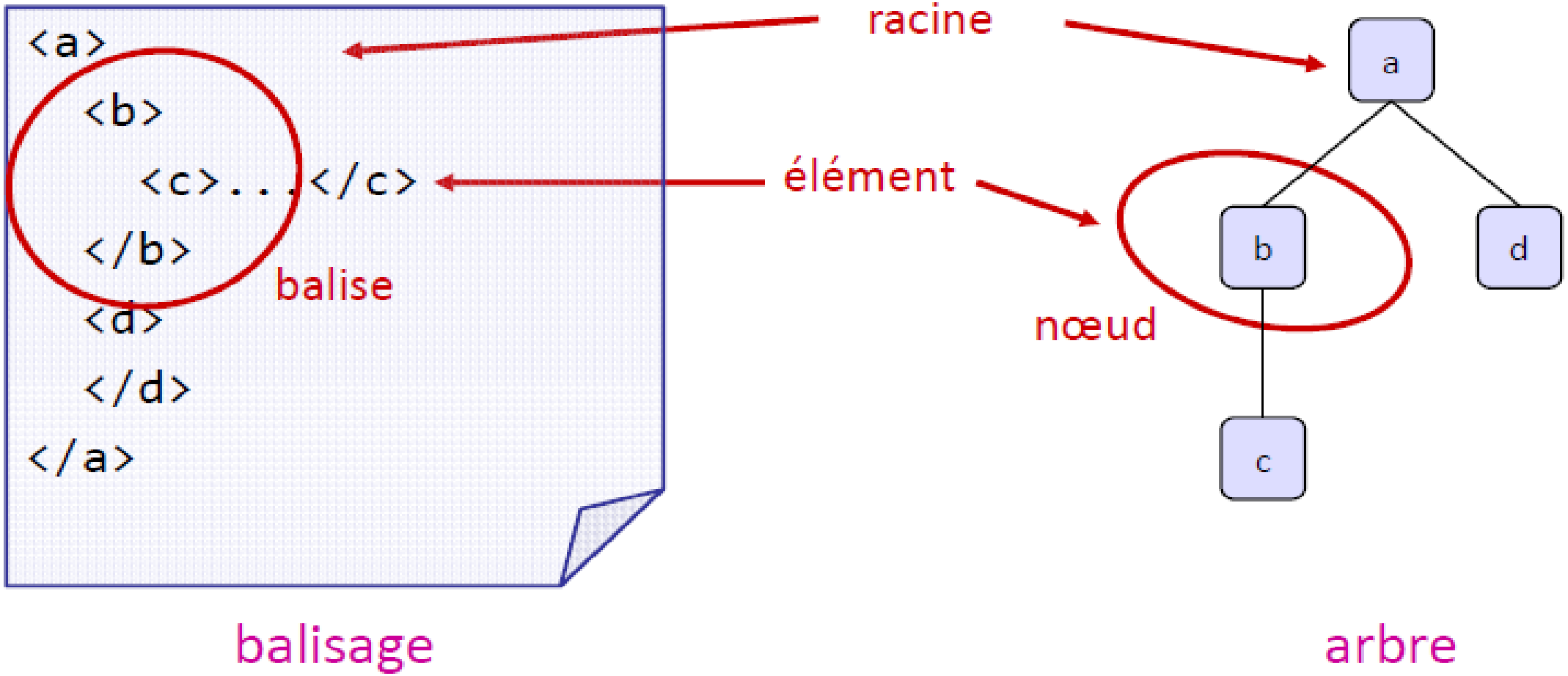
```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Donald Knuth</title>
 </head>
 <body>
  <h1>Donald Knuth</h1>
  <table>
   <tr><td>Date de naissance</td>
       <td>10 janvier 1938</td> </tr>
   <tr><td>Adresse</td>
       <td>Stanford University<br>
           CA 94305<br>
           États-Unis
       </td></tr>
   <tr><td>Profession</td>
       <td>informaticien</td></tr>
  </table>
 </body>
</html>
```

XML

```
<?xml version="1.0" encoding="utf-8" ?>
<personne>
  <prenom>Donald</prenom>
  <nom>Knuth</nom>
  <date_naissance>1938-01-10</date_naissance>
  <adresse>
    <société>Stanford University</société>
    <code_postal type="ZIP">CA 94305</code_postal>
    <pays code = "ISO-3166">US</pays>
  </adresse>
  <profession>informaticien</profession>
</personne>
```

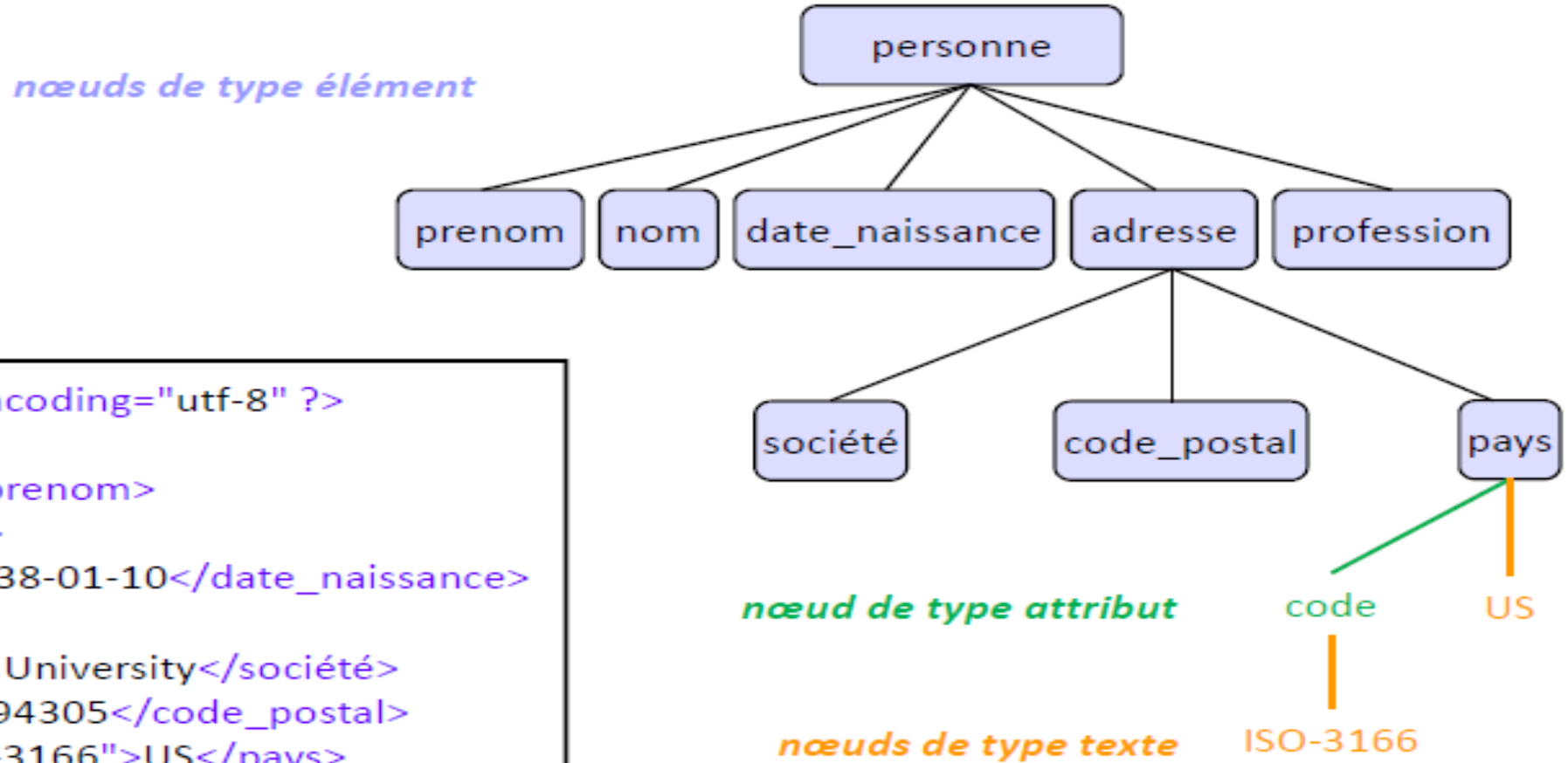
En XML on crée ses propres balises
Structuration logique de données
Pas de représentation envisagée *à priori.*

Ce qu'il faut savoir : dualité texte balisé / arbre



Dualité : Exemple

```
<?xml version="1.0" encoding="utf-8" ?>
<personne>
  <prenom>Donald</prenom>
  <nom>Knuth</nom>
  <date_naissance>1938-01-10</date_naissance>
  <adresse>
    <société>Stanford University</société>
    <code_postal>CA 94305</code_postal>
    <pays code = "ISO-3166">US</pays>
  </adresse>
  <profession>informaticien</profession>
</personne>
```



XML : Syntaxe de base (1/2)

- **Arbre d'éléments :**

  – Balise de fin obligatoire, pas de chevauchement de balises, un seul élément racine

  – Une seule racine ↔ Une seule balise englobe l'ensemble du document

```
<unElement>
        <UnSousElement>
        </UnSousElement>
</unElement>
```

- **Element vide**, i.e. qui ne contient pas de sous-élément

```
<unElementVide />
```

XML : Syntaxe de base (2/2)

- **Contenu**

  <unElement> une donnée textuelle </unElement>

- **Attributs et valeurs**

  <unElement unAttribut= "uneValeur">

  </unElement>

- **Sensibilité à la casse**

  <unElement> </UNELEMENT> ***incorrect!***

- **Commentaire**

  <!-- Note : partie modifiée le 1er avril -->

Nommage des éléments et attributs

Règles de nommage des identifiants :

- Un caractère ou plus
- Caractères autorisés :
  - chiffres
  - lettres (tous alphabets)
  - _ (souligné, *underscore)*
  - - (trait d'union, *hyphen)*
  - . (point, *dot)*
- Sauf 1er caractère : seulement lettres ou _ (souligné)
- Jamais d'espaces, jamais d'autres caractères de ponctuation !

Caractères utilisables dans le contenu

- Presque tous les caractères autorisés...
- Sauf certains caractères spéciaux pour XML, par exemple :
  - pas de " dans un attribut délimité par ""
  - pas de ' dans un attribut délimité par ''
  - < et & absolument interdits
- **Entités caractères :**
  - ' **'**
  - > **>**
  - < **<**
  - " **"**
  - & **&**

Prologue XML = déclaration XML

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

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

<?xml version="1.0" encoding="ISO-8859-1" ?>
```

Rôles du prologue :

– Identifier à coup sûr un fichier XML

– Indiquer la version : 1.0 ou 1.1 (différences négligeables)

– Préciser le codage de caractères utilisé : par défaut jeu de caractères international Unicode, avec le codage UTF-8.

Document XML bien formé

Un document XML est bien formé (*well-formed) s'il* respecte les règles de la syntaxe XML :

- Un document possède une racine et une seule.
- Tout élément a une balise ouvrante et une fermante : <n>...</n>
- Eléments vides : ou
- Les paires de balises ouvrantes/fermantes sont imbriquées : <a>.........
- <a>......... × pas de chevauchement
- Un attribut ne peut apparaître qu'une fois pour un même élément et chaque attribut doit avoir une valeur
- Un processeur XML doit s'arrêter lorsqu'il rencontre une erreur

```
<?xml version="1.0" encoding="UTF-8"?>
<kml>
  <Placemark>
    <name>New York City</name>
    <description>New York City</description>
    <Point>
      <coordinates>-74.006393,40.714172,0</coordinates>
    </Point>
  </Placemark>
</kml>
```

prologue XML
= déclaration XML

Exemple : (Description d'un point géographique en KML – Google Earth)

Exercice

```
1. <?xml version="1.0"?>
2. <!-- this is a note -->
3. <note date=3 janvier>
4. <to>Bob</To>
5. <from>Alice</from>
6. <heading>Reminder</heading>
7. <body>Don't forget me this weekend!</body>
8. </note>
9. <note date="5 janvier" <!-- this is another note --> >
10. <to>Alice</to>
11. <from>Bob
12. <body>No problem & see you soon</body>
13. </note>
14. <note />
```

Observez le document XML suivant :

Questions :

1. Ce document est-il bien formé (i.e. respecte-t-il la syntaxe XML) ?

2. S'il ne l'est pas, corrigez les erreurs.

Document XML valide (1/2)

- Un document peut être valide par rapport à un schéma qui définit un vocabulaire et une structure.
- Un document valide doit respecter des règles définies par le créateur du dit document (en termes de règle d'arborescence, de contenu de balise, etc.) et, en l'absence de règles, le document ne sera jamais considéré comme valide.
- Ceci n'empêche en rien son utilisation, vous pouvez utiliser un fichier non valide... Par contre, cela complexifie considérablement la façon dont vous allez coder son utilisation.

Document XML valide (2/2)

Lorsque vous allez devoir traiter le fichier XML, il va falloir contrôler un certain nombre de choses comme :

- Est-ce que mon nœud <branche> a bien un attribut "nom" de renseigné ?
- Est-ce que ce nœud a bien un ou plusieurs d'autres nœuds ?
- Est-ce que mes nœuds <fruit> ont bien tous des nœuds <nom> et <couleur> ?
- …
- Toutes ces choses seront à faire pour contrôler la validité de la structure que vous manipulez afin d'être sûrs et certains que les données contenues dans celle-ci sont bonnes et utilisables.

Langages de schémas pour XML

Divers langages, divers niveaux d'expressivité :

DTD (Document Type Definition)

- Principalement, définition d'un vocabulaire :
- Noms des balises et attributs

XML Schema

- Vérification plus poussée de la structure
- Notion de type de données

DTD : liaison entre document et DTD : 04 Formes

Reference à un fichier DTD privé :

```
<!DOCTYPE personne SYSTEM
   "http://www.example.com/personne.dtd">
```

Référence à une DTD publique :

```
<!DOCTYPE math PUBLIC "-//W3C//DTD MathML 2.0//EN"
"http://www.w3.org/TR/MathML2/dtd/mathml2.dtd">
```

DTD interne au document :

```
<!DOCTYPE personne [<!ELEMENT personne (prénom, nom)> ...]>
```

Pas de référence à une DTD : <!DOCTYPE html>

Syntaxe et contenu d'une DTD

Syntaxe :

- Syntaxe particulière, mais ressemblant au reste de XML
- Quelques balises prédéfinies de la forme <!BALISE paramètres>
- Pas de déclaration XML dans un fichier .dtd

Contenu principal :

- Déclaration de chaque élément et de son contenu **<!ELEMENT ...>**
- Déclaration des attributs possibles d'un élément **<!ATTLIST ...>**

Déclaration d'un élément

- **Contenu vide** : <!ELEMENT nomÉlément EMPTY>
- **Contenu quelconque** : <!ELEMENT nomÉlément ANY>
- **Contenu textuel (*parsed character data*) :** <!ELEMENT nomÉlément (#PCDATA)>
- ***Contenu composé de sous-éléments***

  par exemple : <!ELEMENT nomÉlément ((fils1,fils2)*|fils3?,fils4+)>

Séquence **,** – Alternative **|**, 0 à N occurrences *****, 1 à N occurrences **+**, 0 ou 1 occurrence **?**

<!ELEMENT adresse (société, boîte_postale?, ville,
(code_postal|code_cedex), pays)>

Déclaration des attributs d'un élément

Un attribut se déclare de cette façon :

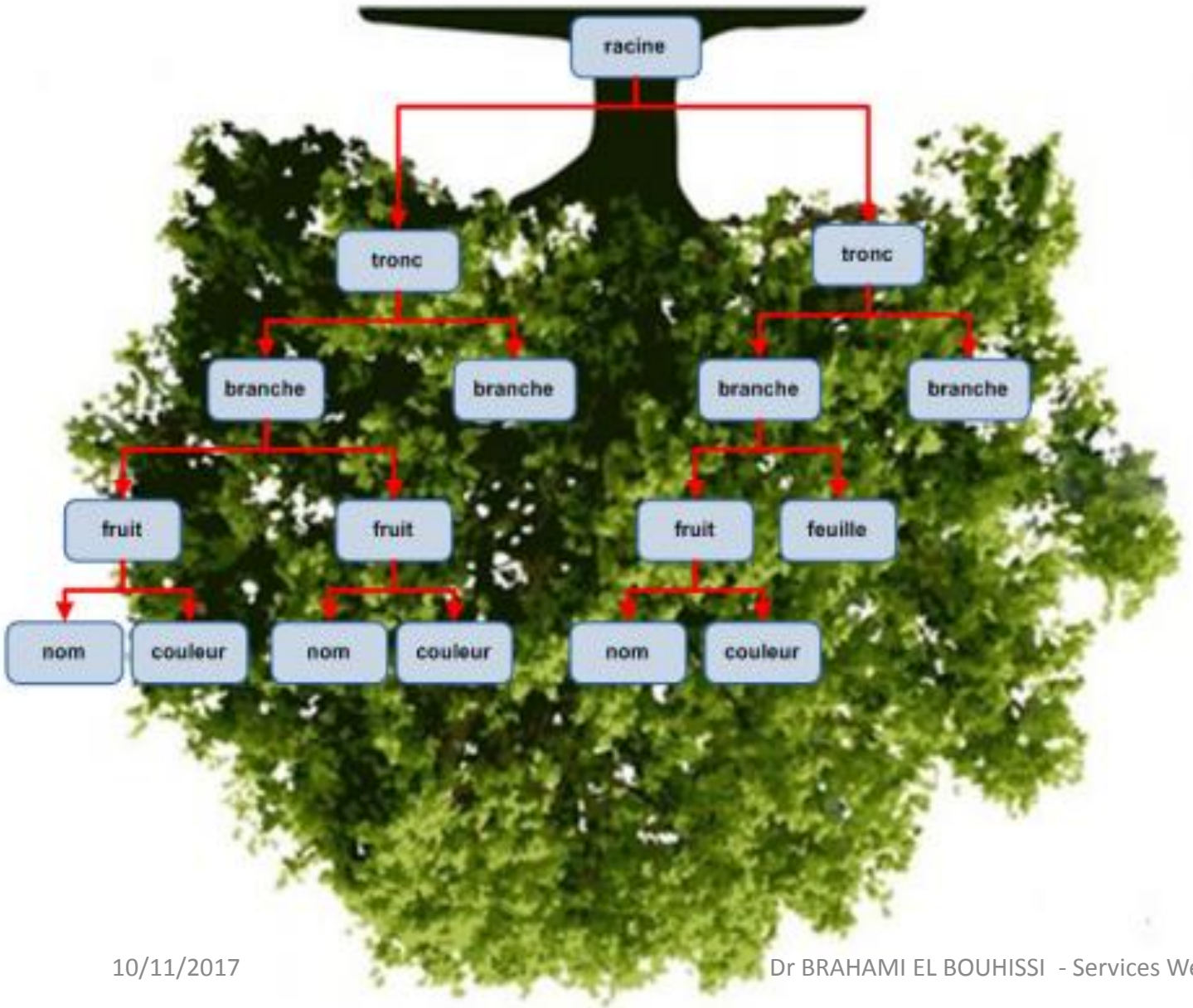
<!ATTLIST balise attribut TypeAttribut TypeDeDefinition ValeurParDefaut>

Il y a trois types de définition :

- #REQUIRED : l'attribut DOIT avoir une valeur, l'attribut est donc obligatoire.
- #IMPLIED : l'attribut est facultatif.
- #FIXED : l'attribut a une valeur fixe, définie à l'avance.

Exemple : <!ATTLIST balise attribut CDATA #IMPLIED > : attribut d'une balise dont le type est chaine de caractère et facultatif

Exemple : DTD d'un arbre




```
<!ELEMENT racine (tronc)+ >
<!ELEMENT tronc (branche)+ >
<!ELEMENT branche (fruit+, feuille+) >
<!ELEMENT fruit (nom,couleur)+ >
<!ELEMENT nom (#PCDATA) >
<!ELEMENT couleur (#PCDATA) >
<!ELEMENT feuille (#PCDATA) >
```

Exercice

Une anthologie est une liste de chansons (au moins une !). Chaque chanson est définie de la manière suivante : elle possède éventuellement un titre, puis elle peut être définie:

- Soit comme une séquence de vers (au moins un),
- Soit comme une succession de couplets/refrains (pouvant éventuellement débuter par un refrain).

Les Couplets et refrains sont des séquences de vers.
Les vers sont simplement du texte, ainsi que le titre.

Questions :

1. Donner la DTD correspondante à cette définition.
2. L'extrait de document suivant est-il bien formé (si non, apportez les corrections nécessaires)?

Valide (si non, apportez les corrections nécessaires)?

```
<anthologie>
<chanson>
<titre> La Javanaise </titre>
<refrain> Ne vous déplaise ... </refrain>
<couplet> J'avoue... </couplet>
<couplet> De vous a moi... </couplet>
</chanson>
<chanson>
<vers> Tralala </vers>
<vers> ohhhhhhhh</vers>
</chanson>
</anthologie>
```

Solution

DTD

```
<!ELEMENT anthologie (chanson+)>
<!ELEMENT chanson ( titre?, ( (vers+) | (refrain?,(couplet,refrain)+)))>
<!ELEMENT titre (#PCDATA) >
<!ELEMENT couplet (vers+) >
<!ELEMENT refrain (vers+) >
<!ELEMENT vers (#PCDATA) >
```

Le document est bien formé puisqu'il n'y a qu'une racine et que les tags sont bien imbriqués de manière correcte. Le document n'est pas valide car non conforme à la DTD:
– couplets et refrain sont composés de vers, et non directement de texte ;
– il manque un refrain après le couplet.

Pour valider un document par rapport à une DTD :

```
java -jar xmlvalidator.jar DTD nom_document.xml
```

(L'outil trouve la DTD tout seul.)

DTD Insuffisant ?

DTD permet d'exprimer des contraintes assez basiques

– Liste des éléments et de leurs attributs

– Règles de structuration des éléments

Mais :

– Impossible de typer réellement les attributs

– Il ne peut y avoir deux éléments de même nom dans deux contextes différents

– Pas d'héritage possible entre éléments : notation lourde

– Pas de prise en compte des espaces de noms

→ Autres langages de schéma plus complets, par ex. XML Schema

XML Schema (XSD)

- **XML Schema** publié comme recommandation par le W3C en mai 2001 est un langage de description de format de document **XML** permettant de définir la structure et le type de contenu d'un document **XML**. Cette définition permet notamment de vérifier la validité de ce document.
- Définit les balises et leurs attributs
- Définit les contraintes de structure des documents
- Espace de noms = préfixe permettant d'éliminer les conflits lorsque plusieurs balises ont des noms identiques, URL (fictive) utilisée comme identifiant
- Les **Schémas XML** offrent plus de possibilités que les DTD.
- Les **Schémas XML** s'écrivent à l'aide d'un langage de type XML.
- Un fichier dans lequel est écrit un **Schéma XML** porte l'extension **".xsd"**.

Apports des schémas XML

- Les **Schémas XML** permettent tout d'abord de *typer* les données. même, il est possible d'aller plus loin en créant nos propres types de données.

- **Les contraintes :** les **Schémas XML** permettent d'être beaucoup plus précis que les DTD lors de l'écriture des différentes contraintes qui régissent un document XML.

- **Des définitions XML :** Un des principaux avantages des **Schémas XML** est qu'ils s'écrivent grâce au XML. Ainsi, pour exploiter un document XML et le Schéma qui lui est associé, vous n'avez en théorie plus besoin de plusieurs outils.

Structure d'un schéma XML

- Bien que les Schémas XML soient écrits avec un langage de type XML, le fichier n'a pas cette extension. Un fichier dans lequel est écrit un Schéma XML porte l'extension "**.xsd**". (Le schéma peut être aussi intégré au fichier XML lui-même.
- La première ligne d'un Schéma XML est :

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

- Comme pour un fichier XML classique, le **corps d'un Schéma XML** est constitué d'un ensemble de **balises.**
- Cependant, une chose ne change pas : la présence d'un **élément racine**, c'est-à-dire la présence d'une balise qui contient toutes les autres. Mais, contrairement à un fichier XML, son nom nous est imposé :

```
<xsd:schema />
```

Exemple

```
<xsd:element name="personne">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="prenom" type="xsd:string"/>
         <xsd:element name="nom" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

```
<personne>
      <prenom>Gaston</prenom>
      <nom>Lagaffe</nom>
</personne>
```

Pour valider un document par rapport à un XML Schema :

java -jar xmlvalidator.jar SCHEMA nom_schema.xsd nom_document.xml

(Il faut explicitement indiquer le schéma.)

Association entre un document et un schéma local

Pour attribuer un schéma de validation local à un document XML, on peut ajouter un attribut situé dans un *namespace spécifique :*

```
<?xml version="1.0"?>
<reference
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="reference.xsd">
<auteur>Bernd Amann et Philippe Rigaux</auteur>
<titre>Comprendre XSLT</titre>
<ISBN>2-84177-148-2</ISBN>
</reference>
```

Association entre un document et un schéma public

Lorsque le schéma est public, mis sur un serveur, c'est un peu différent car il faut définir un *namespace et l'URL d'accès et* il faut ajouter les attributs xmlns et targetNamespace valant le même *namespace à la racine du schéma*

```
<?xml version="1.0"?>
<reference
xmlns="http://www.iut-lannion.fr"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.iut-lannion.fr reference.xsd">
<auteur>Bernd Amann et Philippe Rigaux</auteur>
<titre>Comprendre XSLT</titre>
<ISBN>2-84177-148-2</ISBN>
</reference>
```

Principes généraux des Schémas XML

Comme une DTD, un schéma permet de définir des éléments, leurs attributs et leurs contenus. Mais il y a une notion de typage beaucoup plus forte qu'avec une DTD. Avec un schéma, il faut définir les types de données très précisément :

- Nature des données : chaîne, nombre, date, etc.
- Les contraintes qui portent dessus : domaine de définition, expression régulière, etc.

Avec ces types, on définit les éléments : noms et types des attributs, sous-éléments possibles avec leurs répétitions, les alternatives, etc.

C'est tout cela qui complique beaucoup la lecture d'un schéma.

Structure générale d'un schéma

Un schéma est contenu dans un arbre XML de racine <xsd:schema>. Le contenu du schéma définit les éléments qu'on peut trouver dans le document. Voici un squelette de schéma :

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="itineraire" type="TypeItineraire" />
... définition du type TypeItineraire ...
</xsd:schema>
```

Il valide le document partiel suivant :

```
<?xml version="1.0"?>
<itineraire>
...
</itineraire>
```

Définition d'éléments

Un élément *<nom>contenu</nom> du document est défini par un* élément <xsd:element name="*nom" type="TypeContenu"*> dans le schéma.
Dans l'exemple suivant, le type est xsd:string, c'est du texte quelconque :

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="message" type="xsd:string"/>
</xsd:schema>
```

Ce schéma valide le document suivant :

```
<?xml version="1.0"?>
<message>Tout va bien !</message>
```

Types de données

L'exemple précédent indique que l'élément <message> doit avoir un contenu de type xsd:string, c'est à dire du texte. Ce type est un « type simple ». Il y a de nombreux types simples prédéfinis, dont :

chaîne :

- xsd:string est le type le plus général
- xsd:token vérifie que c'est une chaîne nettoyée des sauts de lignes et espaces d'indentation

date et heure :

- xsd:date correspond à une chaîne au format AAAA-MM-JJ
- xsd:time correspond à HH:MM:SS.s
- xsd:datetime valide AAAA-MM-JJTHH:MM:SS, on doit mettre un T entre la date et l'heure.

Types de données (suite)

Nombres :

- xsd:decimal valide un nombre réel
- xsd:integer valide un entier

De nombreuses variantes comme xsd:nonNegativeInteger, xsd:positiveInteger

autres :

- xsd:ID pour une chaîne identifiante, xsd:IDREF pour une référence à une telle chaîne
- xsd:boolean permet de n'accepter que true, false, 1 et 0 comme valeurs dans le document.
- xsd:anyURI pour valider des URI (URL ou URN).

Restrictions sur les types

Lorsque les types ne sont pas suffisamment contraints et risquent de laisser passer des données fausses, on peut rajouter des contraintes. Elles sont appelées *facettes (facets).* Dans ce cas, on doit définir un type simpleType et lui ajouter des restrictions.

Voici un exemple :

```
<xsd:element name="temperature" type="TypeTemperature" />

<xsd:simpleType name="TypeTemperature">
  <xsd:restriction base="xsd:decimal">
    <xsd:minInclusive value="-30"/>
    <xsd:maxInclusive value="+40.0"/>
  </xsd:restriction>
</xsd:simpleType>
```

Définition de restrictions

La structure d'une restriction est :

```
<xsd:restriction base="type de base">
    <xsd:CONTRAINTE value="PARAMETRE"/>
    ...
</xsd:restriction>
```

```
<xsd:simpleType name="TypeNumeroSecu">
  <xsd:restriction base="xsd:string">
    <xsd:whiteSpace value="collapse"/>
    <xsd:pattern value="[12][0-9]{12}( [0-9]{2})?"/>
  </xsd:restriction>
</xsd:simpleType>
```

Les contraintes qu'on peut mettre dépendent du type de données. Il y a une hiérarchie entre les types qui fait que par exemple le type nombre hérite des restrictions possibles sur les chaînes.

Restriction communes à tous les types

Ces facettes sont communes à tous les types.

longueur de la donnée : xsd:length, xsd:maxLength, xsd:minLength. Ces contraintes vérifient que la donnée

présente dans le document a la bonne longueur.

énumération de valeurs possibles :

```
<xsd:simpleType name="TypeFreinsVélo">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="disque"/>
    <xsd:enumeration value="patins"/>
    <xsd:enumeration value="rétropédalage"/>
  </xsd:restriction>
</xsd:simpleType>
```

Type complexe

Pour modéliser un élément <personne> ayant deux éléments enfants <prénom> et <nom>, il suffit d'écrire ceci :

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="personne" type="TypePersonne"/>
    <xsd:complexType name="TypePersonne">
        <xsd:all>
            <xsd:element name="prénom" type="xsd:string"/>
            <xsd:element name="nom" type="xsd:string"/>
        </xsd:all>
    </xsd:complexType>
</xsd:schema>
```

La structure <xsd:all> contient une liste d'éléments qui doivent se trouver dans le document à valider. Il y a d'autres structures.

Contenu d'un type complexe

- On s'intéresse à ce qu'on met dans un <xsd:complexType>

```
<xsd:complexType name="TypePersonne">
    <xsd:sequence> ou <xsd:choice> ou <xsd:all>...
</xsd:complexType>
```

Les enfants peuvent être :

• <xsd:sequence>éléments. . . </xsd:sequence> : ces éléments doivent arriver dans le même ordre

•<xsd:choice>éléments. . . </xsd:choice> : le document à valider doit contenir l'un des éléments

•<xsd:all>éléments. . . </xsd:all> : le document à valider doit contenir certains de ces éléments et dans l'ordre qu'on veut.

Exemple de séquence

Pour représenter une adresse, les éléments <destinataire>, <rue> et <cpville> doivent se suivre dans cet ordre :

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="adresse" type="TypeAdresse"/>
    <xsd:complexType name="TypeAdresse">
        <xsd:sequence>
            <xsd:element name="destinataire" type="xsd:string"/>
            <xsd:element name="rue" type="xsd:string"/>
            <xsd:element name="cpville" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Nombre de répétitions

Dans le cas de la structure <xsd:sequence>, il est possible de spécifier un nombre de répétition pour chaque sous-élément.

```
<xsd:complexType name="TypePersonne">
    <xsd:sequence>
        <xsd:element name="prénom" type="xsd:string"
                minOccurs="1" maxOccurs="2"/>
        <xsd:element name="nom" type="xsd:string"
                minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
```

Par défaut, les nombres de répétitions min et max sont 1. Pour enlever une limite sur le nombre maximal, il faut écrire maxOccurs="unbounded".

Définition d'attributs

Les attributs se déclarent dans un <xsd:complexType> :

```
<xsd:complexType name="TypePersonne">
    ...
    <xsd:attribute name="NOM" type="TYPE" [OPTIONS]/>
</xsd:complexType>
```

nom	le nom de l'attribut
type	le type de l'attribut, ex: `xsd:string` pour un attribut quelconque
options	mettre `use="required"` si l'attribut est obligatoire, mettre `default="valeur"` s'il y a une valeur par défaut.