Introduction à XML
eXtensible Markup Language

Fondements, modélisation, présentation et programmation

RT3 et GL3 L.A
Ben Sassi Manel
2010/2011

Sommaire du cours

- Fondements de la technologie XML
- Document Type Définition (DTD) et schéma XML
- L'interrogation du Fichier XML : Xpath et XQUERY
- Programmation en utilisant XML
- Application du XML: Les services Web, Ajax

XML: eXtensible Markup Language

Partie 1: fondements et notions de base

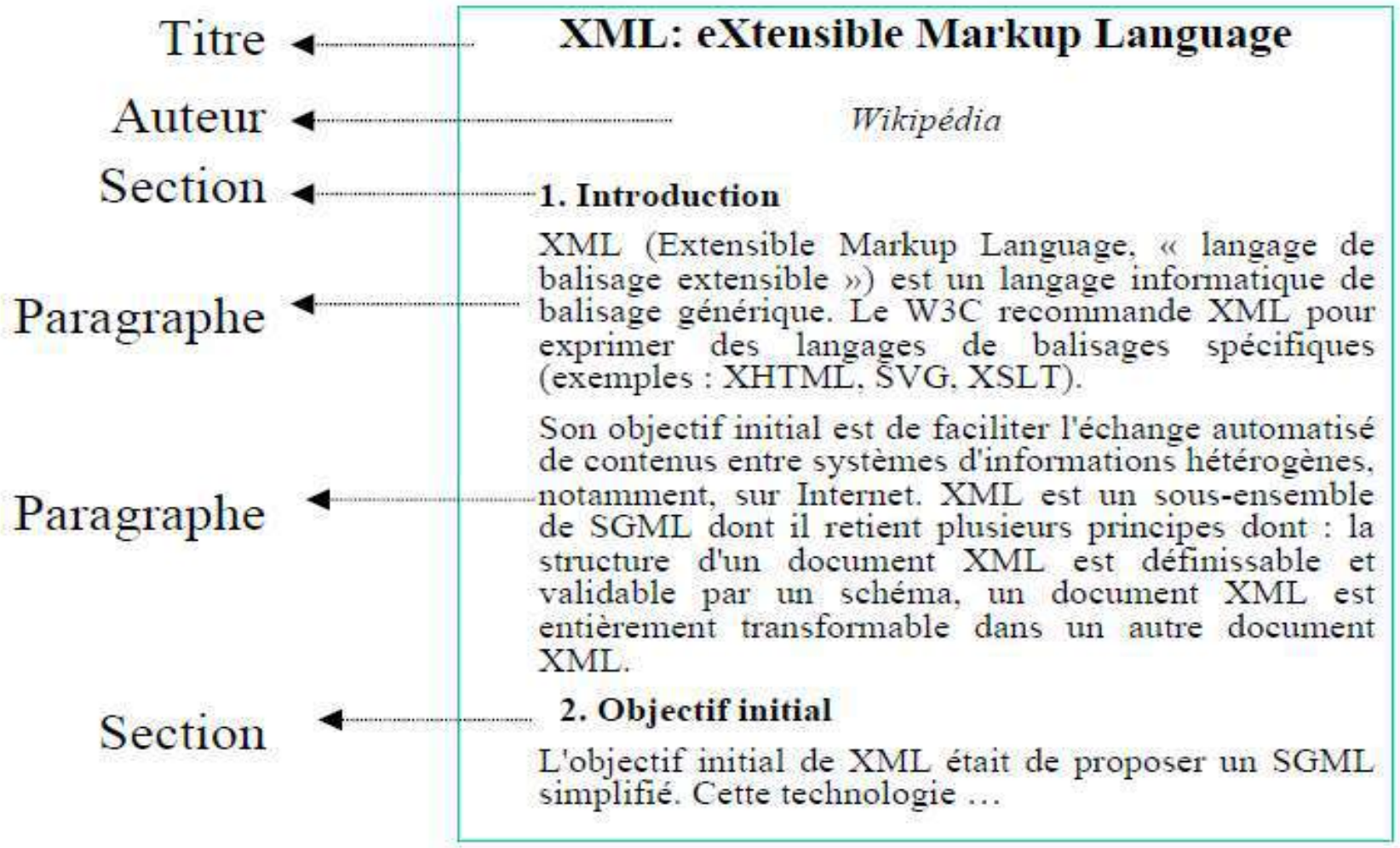
Notion de document électronique

- **Définitions:**
  - Objet qui joue un rôle de médiateur entre les hommes dans le temps et l'espace (échange d'information)
  - Se présente sous la forme de textes, tableaux, dessins, photos, …
  - Un document a un contenu (structure logique) et un fond (structure physique)
  - Un document est un « ensemble formé par un support et une information » (ISO TC-46)

Modèles de document

- **Structure logique**
  - Décrit le contenu d'un document (information)
  - Par exemple: Chapitre, section, paragraphes, figures, notes…
- **Structure physique**
  - Décrit la forme et le support du document (formatage)
  - Par exemple: tomes, pages, cadres, pavés, fenêtres

Exemple d'un document



Exemple d'un document: Structure Logique

<Article>

<Titre> XML: eXtensible Markup Language </Titre>

<Auteur>Wikipédia</Auteur>

<Section titre = "Introduction">

<Paragraphe>XML (Extensible Markup Language, « langage de balisage extensible ») est un langage informatique de balisage générique. Le W3C recommande XML pour exprimer des langages de balisages spécifiques (exemples : XHTML, SVG, XSLT). …
</Paragraphe>

<Paragraphe> Son objectif initial est de faciliter l'échange automatisé de contenus entre systèmes d'informations hétérogènes, notamment, sur Internet. XML est un sous-ensemble de SGML dont il retient plusieurs principes dont : la structure d'un document XML est définissable et validable par un schéma, un document XML est entièrement transformable dans un autre document XML. </Paragraphe>

</Section>

<Section titre= "Objectif initial"> …

</Section>

</article>

→**Reflète le contenu du document!!**

Exemple d'un document: Structure Physique

```
<Article>
<Titre police="Times"  taille="24" position="centré" format="gras"/>
<Auteur police="Times"  taille="20" position="centré"
   format="italique"/>
<Section numero="1" police="Times"  taille="18"
    position="centré" format=" gras "/>
<Paragraphe police="Times"  taille="18" position="justifié"/>
</article>
```

→C'est la mise en forme du document!!

Langages de représentation de documents

- **Par ordre chronologique:**
  - **SGML** (Norme ISO 8879 en 1986, révisée en 1988 et 1994)
    - Méta-langage général
  - **HTML** (Standard W3C depuis 1989)
    - Structure générale figée
  - **XML** (Standard W3C depuis 1998)
    - Méta-langage simplifié compatible SGML

SGML: brève présentation

- Une norme internationale :
  - **S**tandard **G**eneralized **M**arkup **L**anguage
  - ISO 8879 - 1989
- Un **méta langage** de balisage de documents
  - lisible par l'être humain et traitable par une machine
  - permet de définir des langages de balisage
- Les documents sont balisés conformément à la grammaire (la *DTD: Document Type Definition*)
  - instances de DTD
  - permet un balisage sémantique du fond.
- Implique la notion de validité d'un document

SGML: Objectifs

- Séparation du fond et de la forme
  - possibilité de multiples présentations
  - un seul document en SGML (*regroupant les deux structures*)
  - plusieurs formats : Postcript, HTML, etc.
- Support de traitements sur le contenu des documents sans prise en compte de la forme
- Format de stockage et d'échange normalisé

SGML: Critiques

- Un langage professionnel, concis et abstrait → *Très lourd et complexe pour la mise en œuvre de documents respectant ce format*
- Une grande rigueur est demandée à l'entrée des documents
- Standard complexe et complet pour le traitement des documents
- Liens hypertextes possibles mais complexes

HTML : présentation

- ***HyperText Markup Language***, Proposé par le W3C comme format de documents sur le Web
- Langage simple avec des balises standardisées permettant la mise en forme d'un texte.
- Standard reconnu par tous les navigateurs.
- Langage très populaire sur le Web!!

```
<HTML>
<HEAD>
<TITLE> Exemple </TITLE>
</HEAD>
<BODY>
<H1>Contenu du document</H1>
<A HREF = "http://www.server.fr/Info /dir/test.html"> une référence externe</A>
</BODY>
</HTML>
```

HTML : inconvénients

- **Normalisation des différentes balises est difficile**
  - Les constructeurs ont eu tendance à définir leurs propres balises pour répondre à leurs besoins (incompatibilité)
  - HTML est dédié pour un seul type de terminaux
- **Mises à jour difficiles**
  - Restructuration ou remise en forme de l'ensemble des pages du site est un travail fastidieux !!
  - Incapacité d'extension sans "plugins" coté client (formules mathématiques, modélisations de molécules, scènes 3D…).
- **Mélange de structures logique et physique**
  - Données utiles mélangées avec la mise en forme
  - Difficultés à trouver l'information recherchée!!

SGML et HTML : Résumé

SGML	HTML
➢ Langage puissant pouvant décrire toute structure	➢ Spécialisation de SGML
➢ Documents difficile à définir	➢ Adapté à la présentation
➢ Documents difficiles à utiliser	➢ Inadapté à l'échange entre programmes

XML : Présentation

- *XML*=>*eXtensible Markup Language* , un nouveau langage d'échange basé sur le balisage
- *XML*=> un sous ensemble de SGML, mais plus simple que SGML!

- *XML*=> plus ouvert que HTML
- *XML* => fut développé en 1996, et standardisé par le W3C en 1998.
- *XML* est l'un des membres d'une grande et grandissante famille de langages connexes et coopérant qui inclue: DTD, XSL, XSTL, CSS, XPath, XPointers, XLinks, XML Schema…

XML: objectifs fixés par le W3C (1)

- XML doit pouvoir être utilisé sans difficulté sur Internet → ***simplicité ??!!***
- XML doit soutenir une grande variété d'applications → ***portabilité??***
- XML doit être compatible avec SGML et HTML
- Il doit être facile d'écrire des programmes traitant les documents XML → ***Facile??***
- Le nombre d'options dans XML doit être réduit au minimum, idéalement à aucune.
- Les documents XML doivent être lisibles par l'homme.

XML: objectifs fixés par le W3C (2)

- Les documents XML doivent être raisonnablement clairs
- La spécification de XML doit être disponible rapidement
- La conception de XML doit être formelle et concise
- Il doit être facile de créer des documents XML

Les points forts de XML

- Séparation de la structure et de la présentation
- Moins confus que HTML
- Plus simple que SGML
- Idéal pour l'échange de données semi-structurées
- Utilisable entre machines hétérogènes

XML: utilités (1)

- XML est un *Méta-langage universel* pour représenter les données échangées sur le Web, qui permet au développeur de délivrer du contenu depuis les applications à d'autres applications ou aux navigateurs.
- XML standardise la manière dont l'information est :
  - Échangée: échange facile des données entre les applications
  - Présentée: séparation contenu et forme
  - Archivée: dans des fichiers structurées
  - Retrouvée: indexer l'information d'une manière à la trouver aisément
  - Cryptée: sécurité des informations

XML: utilités (2)

- **Définir vos propres langages d'échange**
  - Commande, facture, bordereau de livraison, etc.
- **Modéliser des documents et des messages**
  - Modèle logique de données
  - Eléments typés agrégés (DTD, XML Schema)
- **Publier des informations**
  - Neutre du point de vue format
  - Mise en forme avec des feuilles de style
- **Archiver des données**
  - Auto-description des archives (recherche d'information)

Concepts de XML

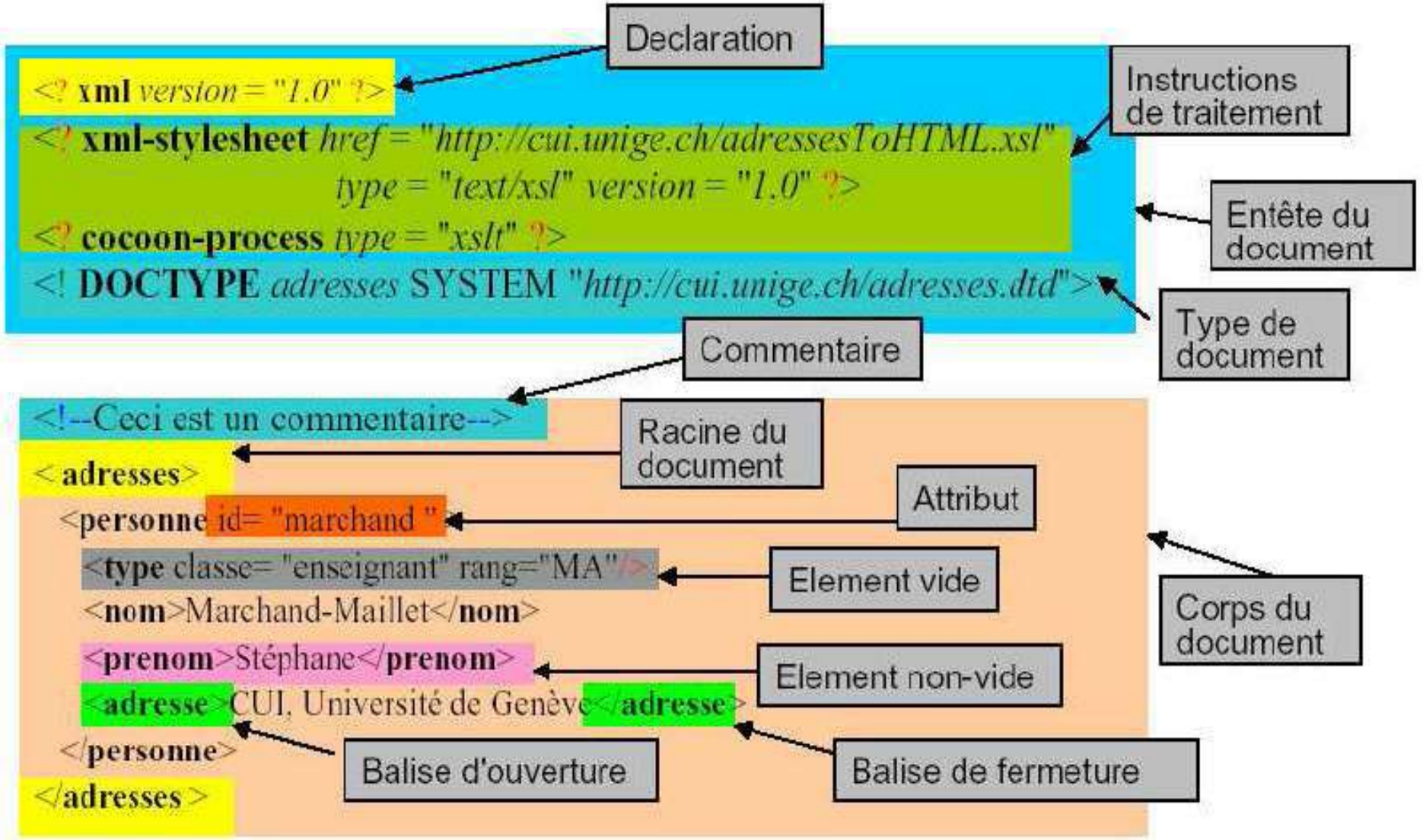
- **Balise (ou tag ou label)**
  - Marque de début et fin permettant de repérer un élément textuel
  - Forme: <balise> de début, </balise> de fin
- • **Elément de données**
  - Texte encadré par une balise de début et une de fin
  - Les éléments de données peuvent être imbriquées

```
<producteur>
        <adresse>
                <rue>A. Einstein</rue>
                <ville>Villeurbanne</ville>
        </adresse>
</producteur>
```

- **Attribut**
  - Doublet nom="valeur" qualifiant une balise

```
<producteur no="160017" region="Rhône">
```

Exemple 1 d'un document XML

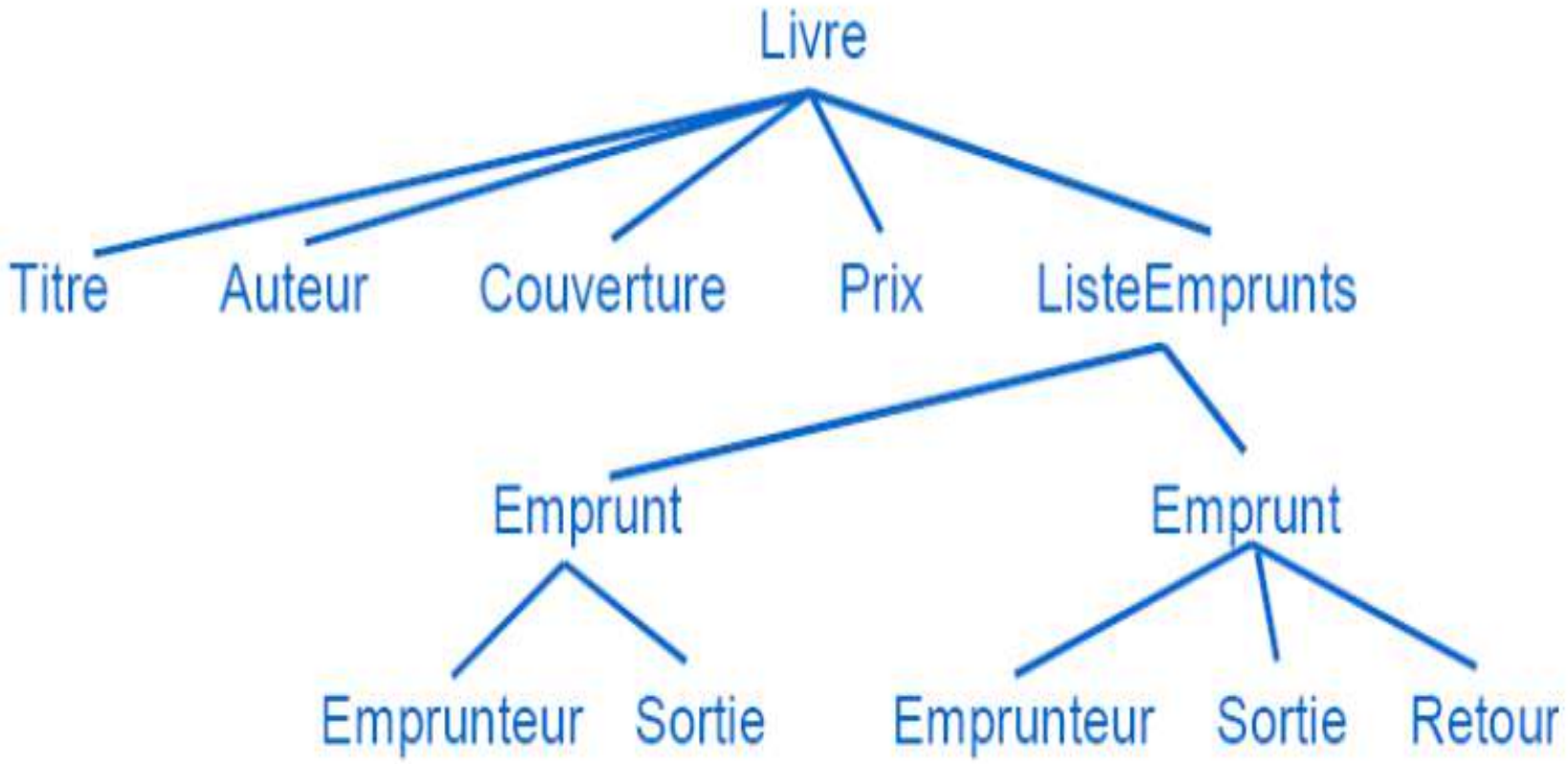


Exemple 2 d'un document XML

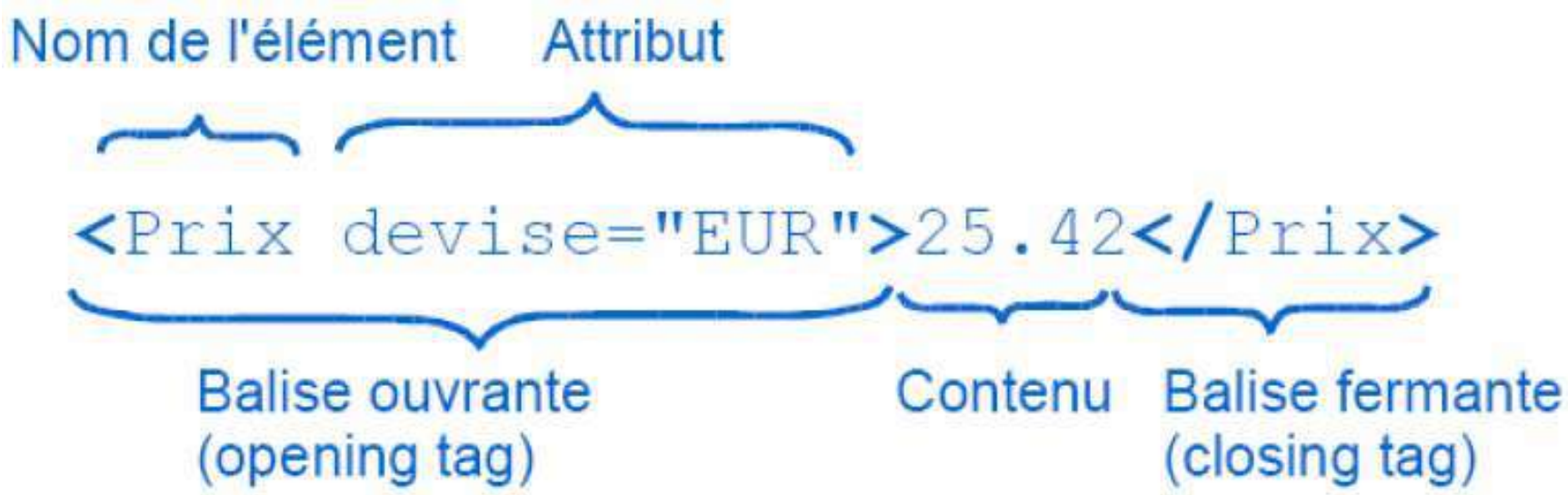
```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<?xml-stylesheet href="bib.css" ?>
<?cocoon-process type="xslt" ?>
<!DOCTYPE Livre SYSTEM "Livre.dtd">

<Livre>
  <Titre>Les réseaux</Titre>
  <Auteur>A. Tanenbaum</Auteur>
  <Couverture imgsrc="/imgs/res-tan.jpg"/>
  <Prix devise="EUR">25.42</Prix>
  <ListeEmprunts>
    <Emprunt>
      <Emprunteur>François Duchemin</Emprunteur>
      <Sortie>25/09/2000</Sortie>
      <Retour>02/10/2000</Retour>
    </Emprunt>
    <Emprunt>
      <Emprunteur>Hervé Delarue</Emprunteur>
      <Sortie>05/10/2000</Sortie>
    </Emprunt>
  </ListeEmprunts>
</Livre>
```

Structure Arborescente d'un document XML



La structure d'un document XML



- ***Quand utiliser les attributs?***
  - Valeur unique de type simple (information monovaluée)
- ***Quand utiliser les éléments?***
  - Valeur de type complexe (énumérations, possède des propriétés)

La syntaxe du XML (1)

- Le XML est un langage de balises [Markup Language]
  - Nous pouvons définir nos propres balises!! Ok Mais, il faut respecter les règles suivantes (dérivées du JavaScript):
  - 1. Les noms peuvent contenir des lettres, des chiffres ou d'autres caractères
  - 2. Les noms ne peuvent débuter par un nombre ou un signe de ponctuation.
  - 3. Les noms ne peuvent commencer par les lettres xml (ou XML ou Xml…).
  - 4. Les noms ne peuvent contenir des espaces

La syntaxe du XML (2)

- 5. La longueur des noms est libre mais on conseille de rester raisonnable.
- 6. On évitera certains signes qui pourraient selon les logiciels, prêter à confusion comme "-", ";", ".", "<", ">", etc.
- 7. Les caractères spéciaux pour nous francophones comme é, à, ê, ï, ù sont à priori permis mais pourraient être mal interprétés par certains programmes.
- 8. Les balises sont sensibles au majuscules et minuscules [case sensitive] exemple: ***<Message> != <message>***
- 9.Toute balise ouverte doit impérativement être fermée. Exemple: Point 1 ou <image/>

La syntaxe du XML (3)

- 10. Les balises doivent être correctement imbriquées.

Exemple: <*parent*><*enfant*>*Marine*</*enfant*></*parent*>

- →des balises mal imbriquées sont des fautes graves de sens: <parent><enfant>Loïc</parent></enfant> incorrecte!!

- 11. Tout document XML doit comporter une racine.
- 12. Les valeurs des attributs doivent toujours être mises entre des guillemets. Exemple: <*date anniversaire*="*071185*">

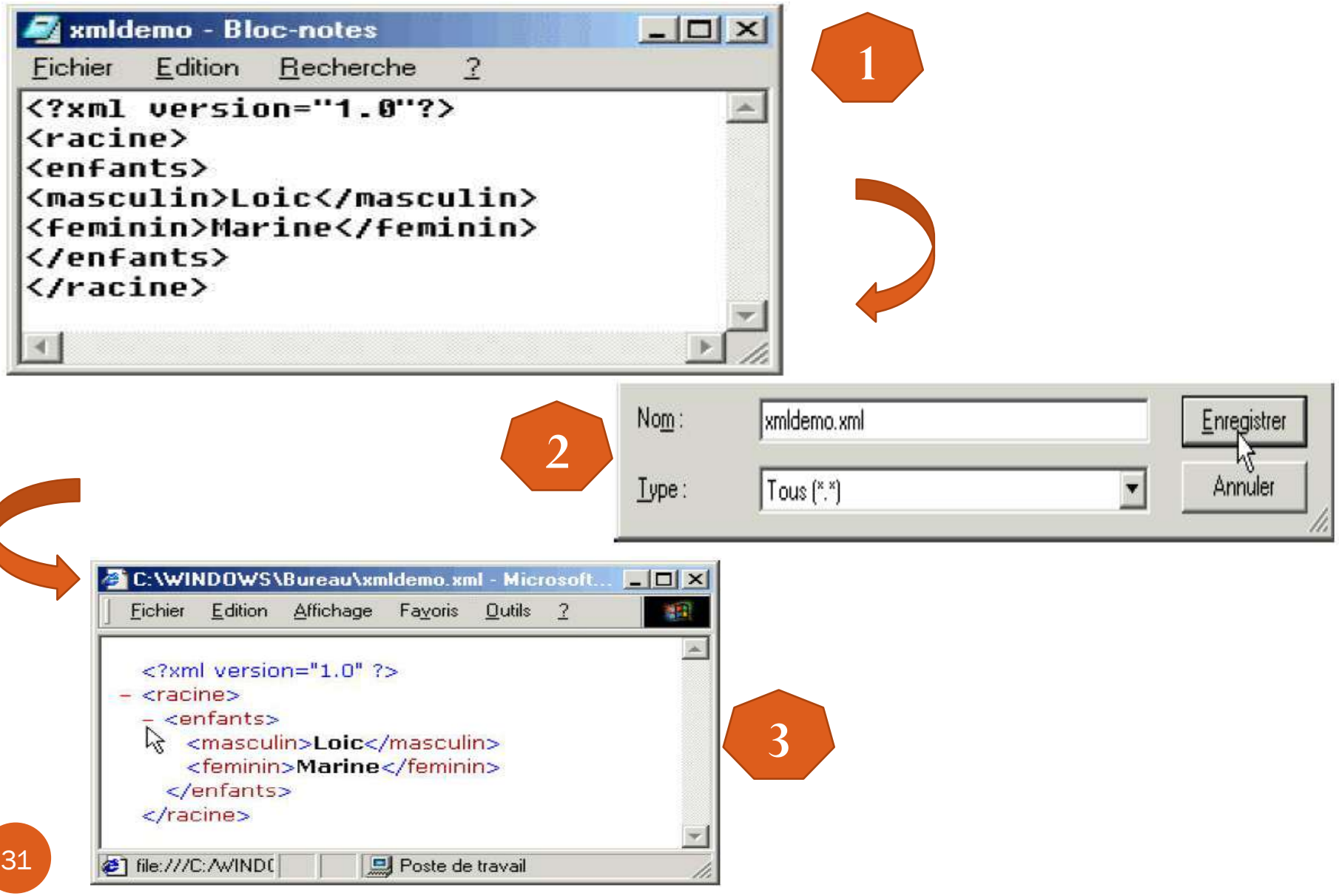
Votre document doit respecter la syntaxe du XML!!

Votre document est bien formé

Un premier exemple (1)

- **<?xml version="1.0" encoding="ISO-8859-1"?>**
  - La déclaration <?xml version="1.0"?> indique au navigateur que ce qui suit est un document XML selon sa version 1.0. Le jeu de caractères "ISO-8859-1" est pour nous francophones. Il existe d'autres encodages, comme UTF8, …
- **<racine>**
  - L'élément racine indispensable au XML.
- **… suite du document XML (bien formé!)…**
- **</racine>**
  - Le document XML se termine obligatoirement à la fermeture de la balise de racine.

Un premier exemple (2)



Récapitulons: XML

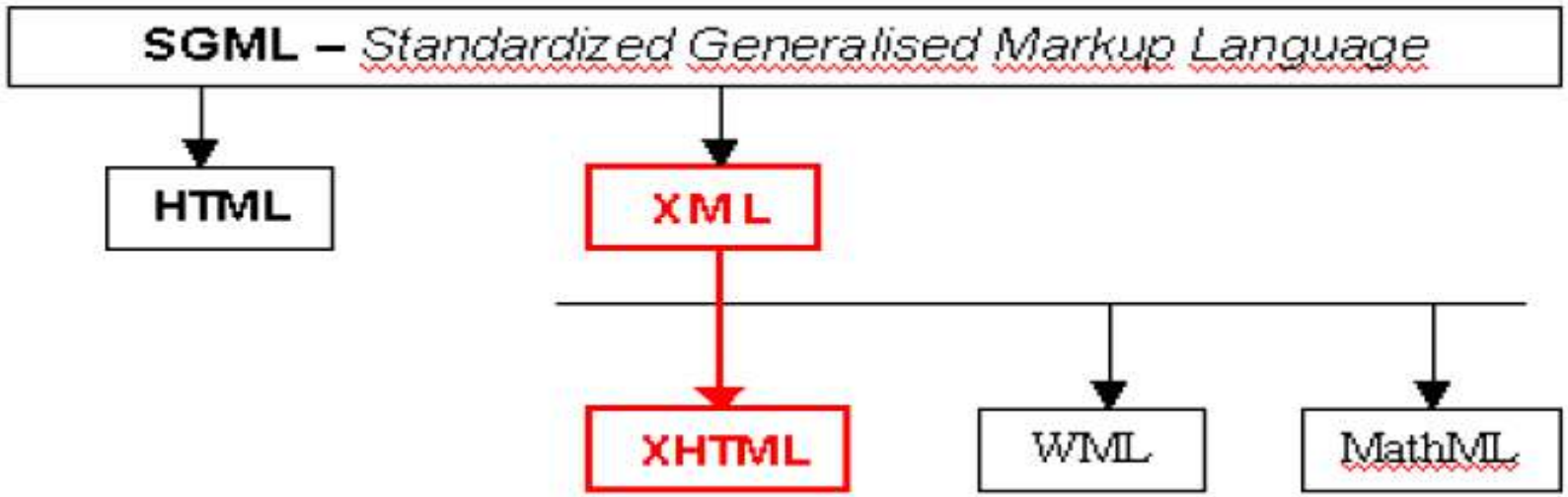
- Xml est un 'format' d'échange entre les applications
  - Il permet de définir des « langages» pour décrire des «données» → c'est un métalangage
  - Permet de représenter pratiquement toute information semi-structurée → il peut jouer le rôle d'une base de données
  - Offre de nombreux outils d'analyse, *parsing, des standards pour interroger et transformer les document XML*, …

→Un standard de « facto » bien adapté au web!!

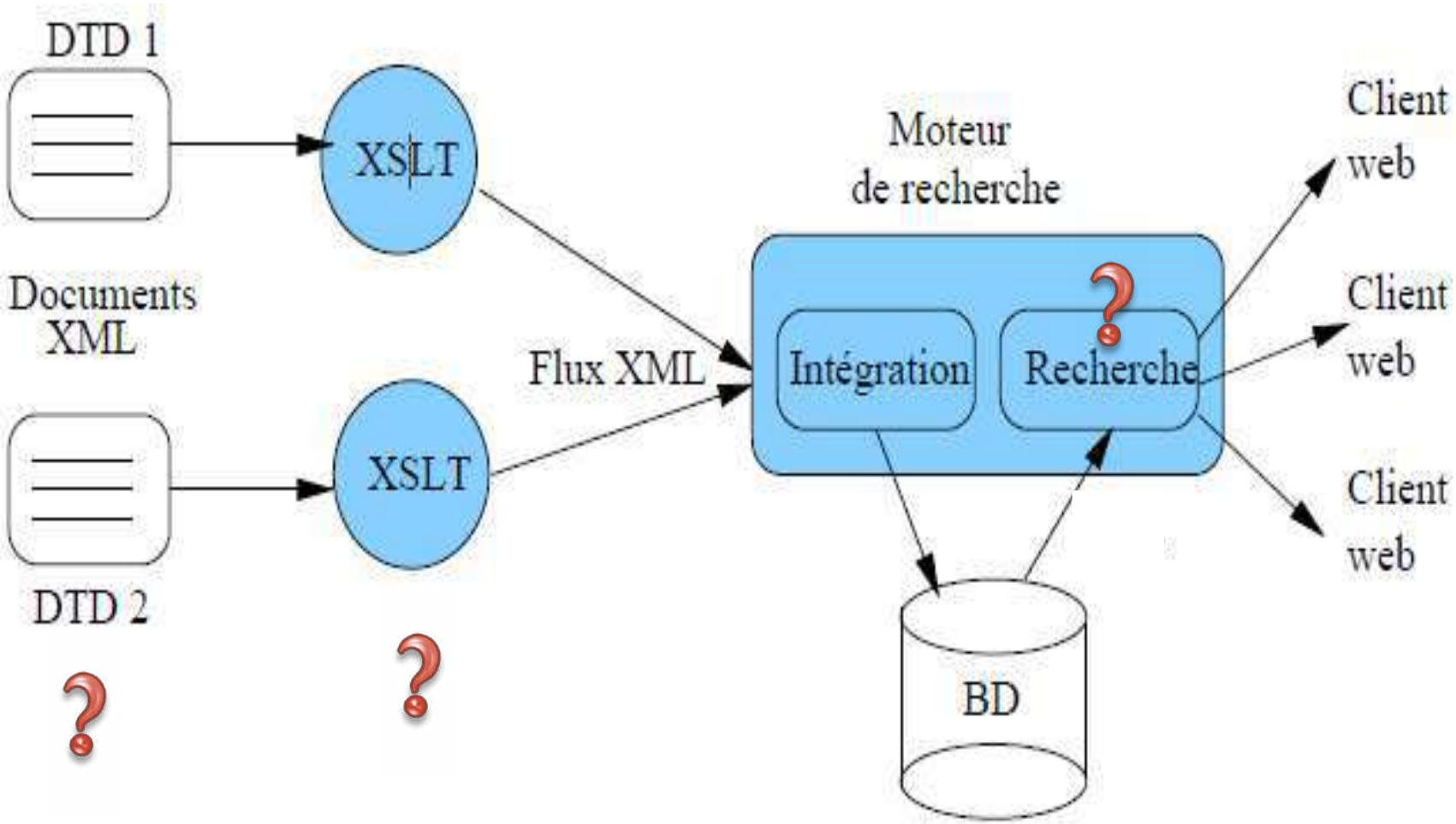
Récapitulons: XML , HTML et XHTML

- XHTML est le successeur du Html mais aussi du XML
  →un ménage ??

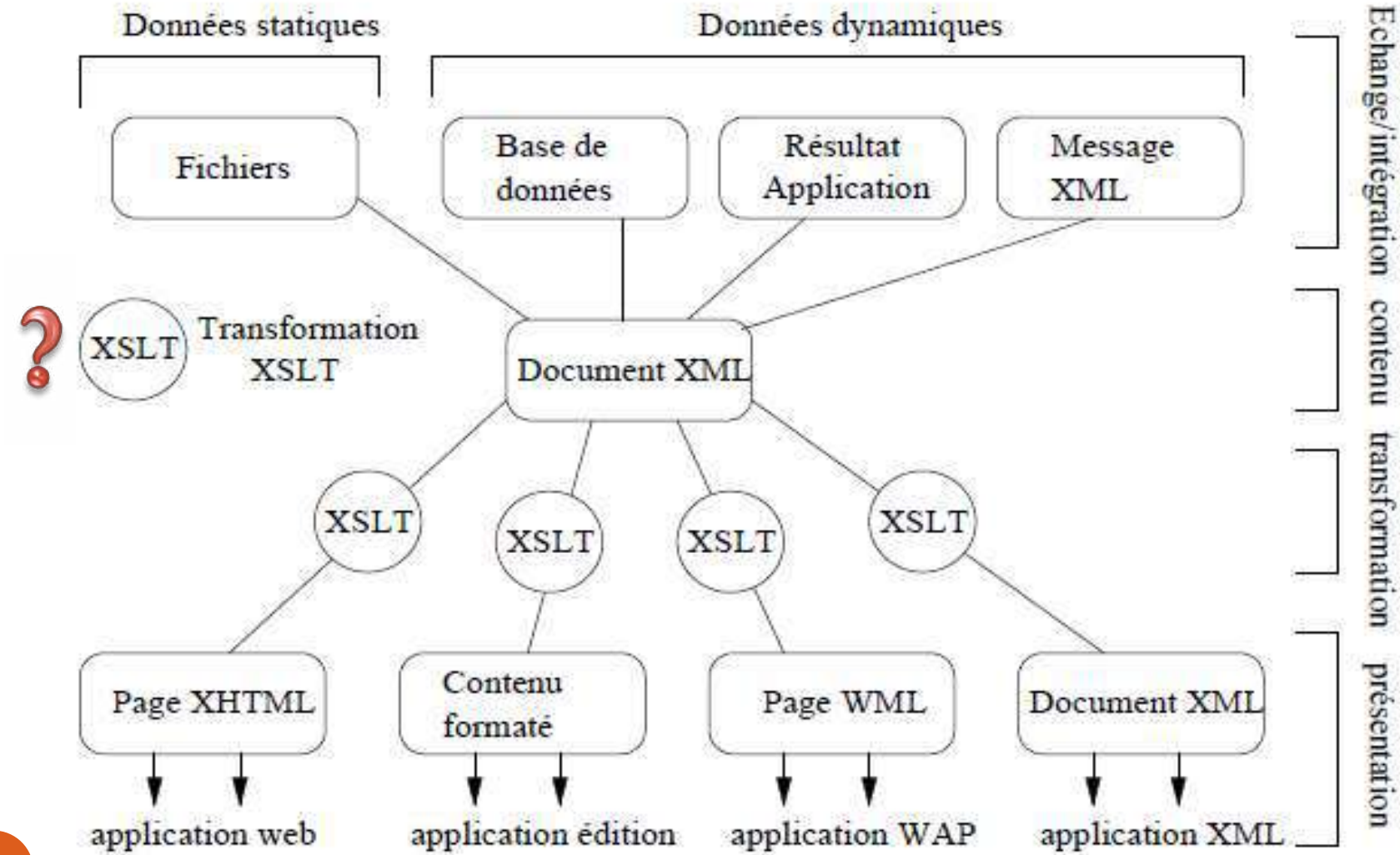
- **→ c'est une reformulation du HTML 4.0 selon la syntaxe et les règles du XML.**



Exemple d'architecture d'application



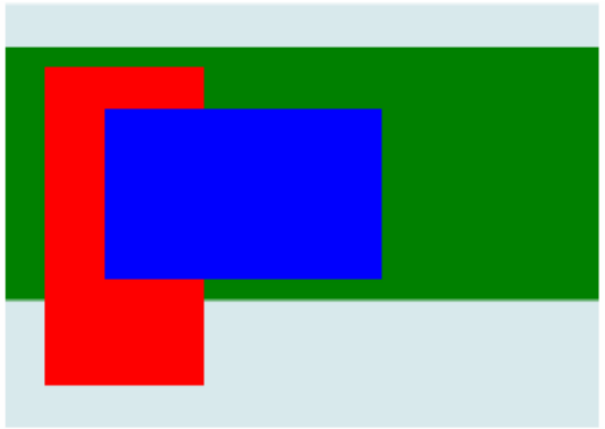
Gestion de l'information avec XML



XML: contextes d'utilisation

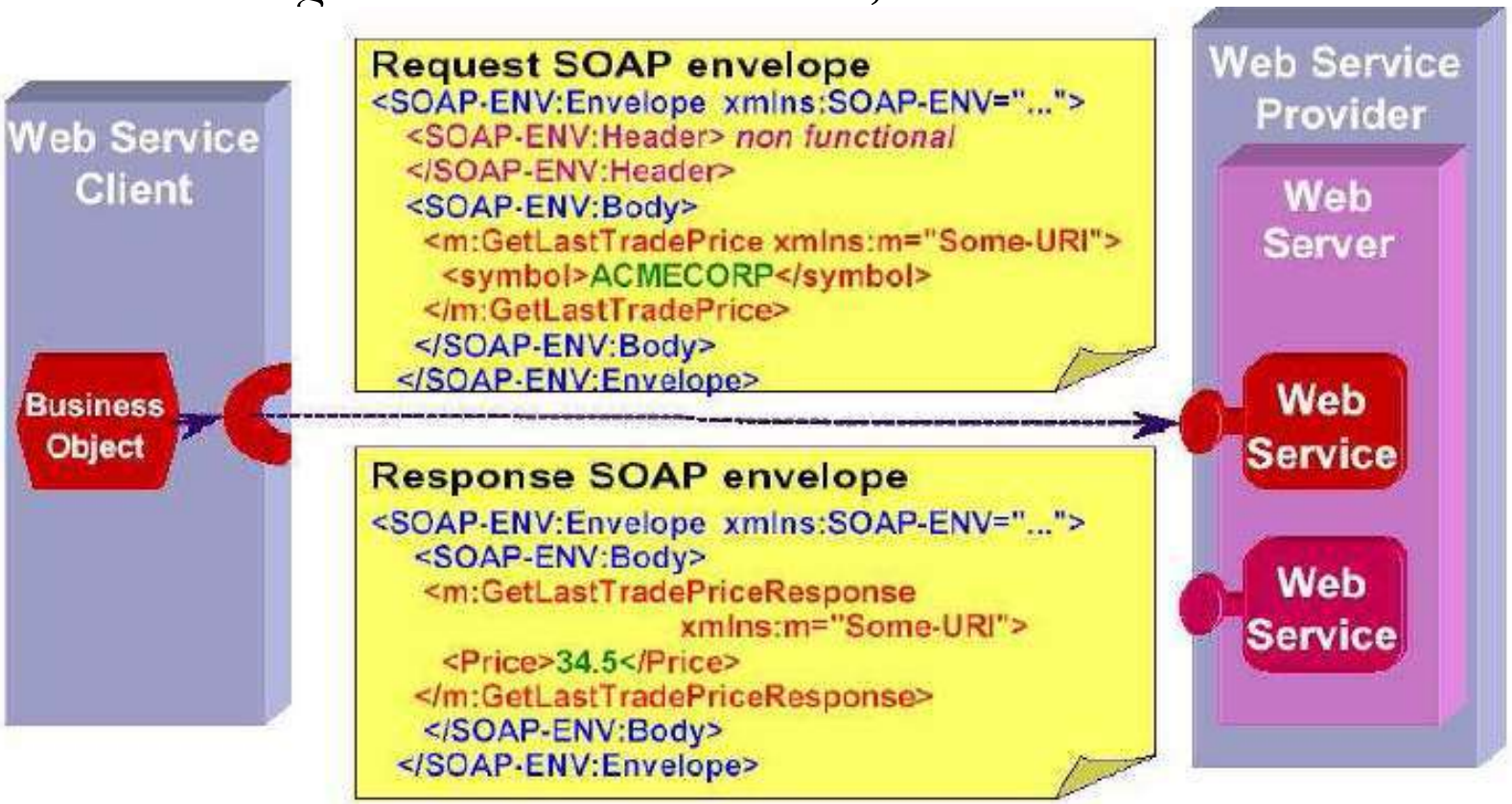
- Stockage de données
  - Format spécifiques (exemple: SVG)

```
<?xml version="1.0" encoding="utf-8"?>
<svg
   xmlns="http://www.w3.org/2000/svg"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:ev="http://www.w3.org/2001/xml-events"
   version="1.1"
   baseProfile="full"
   x="0"
   y="0"
   width="300"
   height="200"
   id="svg2">
  <title>Rectangles</title>
  <defs
     id="defs4" />
  <g
     id="layer1">
    <rect
       width="300"
       height="120"
       x="0"
       y="20"
       fill="green"
       id="rect1306" />
    <rect
       width="80"
       height="150"
       x="20"
       y="30"
       fill="red"
       id="rect1308" />
    <rect
       width="140"
       height="80"
       x="50"
       y="50"
       fill="blue"
       id="rect1310" />
  </g>
</svg>
```



XML: contextes d'utilisation (2)

- Echange de données
  - Échanges Business to Business, services web…



Bases de données vs XML

Approche « Donnée »	Approche « Document »
Structuration forte et simple	Structuration faible et complexe
Compatibilité SGBDR existants	Systèmes documentaires spécialisés
Mise à jour en place	Gestion de versions
Intégrité sémantique	Recherche textuelle
Indexation exacte	Indexation approchée
Adapté au transactionnel et décisionnel	Accès type moteur de recherche
Performances attendues «moyenne » à « forte » pour une volumétrie « moyenne »	Performances attendues « moyenne » pour une volumétrie « forte »

Vous pouvez faire maintenant l'équivalence!!

Passons aux Exercices!

XML: eXtensible Markup Language

Partie 2: Modélisation des documents

Document Type Définition (DTD) et schéma XML

Document Type Définition (DTD)

- **DTD (*Document Type Definition*)**
  - termes équivalents : type de document, classe de document
- *→Une DTD ou/et un schéma permettent de définir son propre langage basé sur XML à savoir leVocabulaire (balises) et la Grammaire (imbrications)*
- **Exemple:**
  - Un document correspondant à une œuvre littéraire comporte:
    - Un nom d'auteur et plusieurs ouvrages
    - Chaque ouvrage a un titre et s'étend éventuellement sur plusieurs tomes numérotés
    - Chaque tome est divisé en chapitre
    - Chaque chapitre a un titre et est divisé en paragraphes
    - Chaque paragraphes comporte du texte

Document Type Définition (DTD)

- **Validité des documents:**
- ***Document bien formé (Well Formed document)***
  - Guillemets (ou apostrophes) obligatoires autour des valeurs exemple *<adresse id='2' type='domicile' />*
  - Les éléments vides utilisent une notation spécifique exemple: *<image src='image3.gif' /> <image src='image3.gif'></image>*
  - Les balises doivent être correctement imbriquées, exemple: **NON ! **
  - Le document a une seule racine
  - Un attribut est unique dans son élément
- ***Document valide (Valid document)***
  - := bien formé + conforme à la DTD ou au schéma qui lui est associé c.à.d toutes les balises utilisées sont définies dans la DTD et elles sont utilisées dans l'**ordre spécifié dans la DTD**

Exercice: documents bien formés?

- QUESTION: Est-ce que les documents suivants sont bien formés ?
- 1er document:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<titre>Ma vie</titre>
<sous-titre>par Christian Rémillard</sous-titre>
<para>Il était une fois...</para>
```

- 2ème document

```
<comité>
  <nom>Comité pour la révision des métadonnées</nom>
  <membres nom="Louis Dantin" nom="Charles Gil" nom="E. L. Massicotte"/>
</comité>
```

- 3ème document

```
<comité>
  <nom>Comité pour la révision des métadonnées</nom>
  <membres>
    <membre nom="Louis Dantin"/>
    <membre nom="Charles Gil"/>
    <membre nom="E. L. Massicotte"/>
  </membres>
</comité>
```

Document Type Définition (DTD)

- On utilise la notion de la DTD lorsqu'on utilise de façon récurrente des balises dans un document XML: ***exemple*** , *on a plusieurs auteurs pour un livre* ☺
- Deux manières pour inclure une DTD à un document XML*: DTD* ***interne*** *au document et* ***externe*** *au document XML*
- **La DTD interne**
- On peut inclure son propre DTD au code source du fichier XML.
- *Syntaxe:*

```
<!DOCTYPE element-racine [
   declaration des éléments
]>
```

Document Type Définition (DTD)

→ Prenons un 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>
```

Début du DTD interne avec parent comme élément de racine.

Début du DTD interne avec parent comme élément de racine.

La racine «parent» contiendra les sous éléments «garcon» et « fille ».

#PCDATA indique au Parser XML que l'élément garçon contient des données exprimées en chiffres ou en lettres.

Fin de la DTD

Document Type Définition (DTD)

- **La DTD externe**

La DTD externe suivra la syntaxe suivante :

```
<!DOCTYPE élément-racine SYSTEM
"nom_du_fichier.dtd">
```

→Le même fichier XML

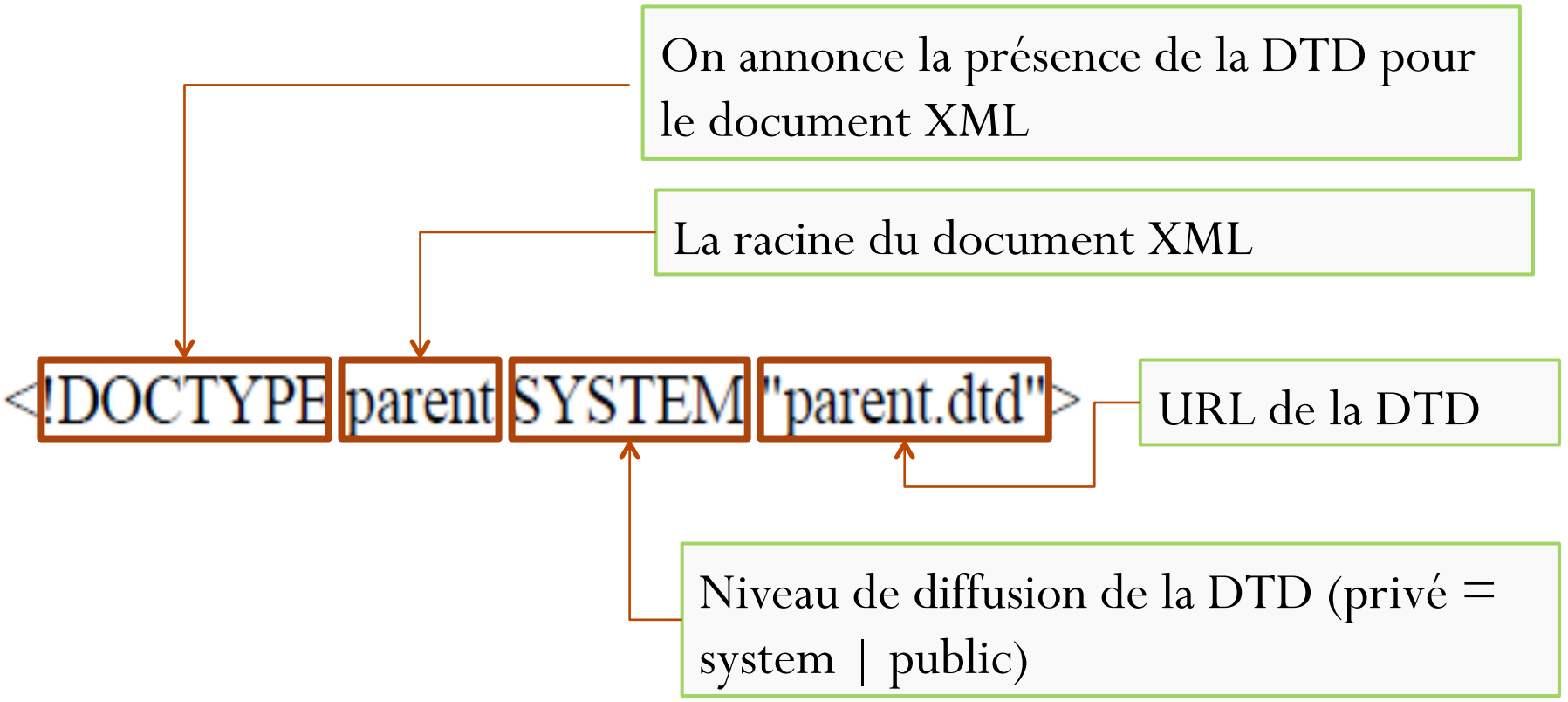
```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE parent SYSTEM "parent.dtd">
<parent>
<garcon>Loic</garcon>
<fille>Marine</fille>
</parent>
```

→ DTD externe correspondante

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

Document Type Définition (DTD)

La déclaration de la DTD externe au niveau du document XML



Document Type Définition (DTD)

- **Déclaration d'élément simple**

<! ELEMENT balise (définition) >

→*Le paramètre définition représente soit un type de donnée prédéfini, soit un élément de données composé, constitué lui même d'éléments*

- *Types prédéfinis* (*Types de base (= type des données entre <nomBalise> et </nomBalise>)*)
  - *ANY* : L'élément peut contenir tout type de donnée et/ou balise
  - *EMPTY* : L'élément ne contient pas de données spécifiques
  - *#PCDATA* : L'élément doit contenir une chaîne de caractères
- *Exemple*
  - <! ELEMENT Nom (#PCDATA)>
  - <Nom>Victor Hugo</Nom>

Document Type Définition (DTD)

- **Déclaration d'élément composé**
  - Définit une séquence ou un choix d'éléments
  - Syntaxe spécifique avec opérateurs de composition d'éléments :

<! ELEMENT balise (composition) >

Opérateur	Signification	Exemple
+	L'élément doit être présent au minimum une fois	A+
*	L'élément peut être présent plusieurs fois (ou aucune)	A*
?	L'élément peut être optionnellement présent	A?
|	L'élément A **ou** B peuvent être présents (pas les deux)	A|B
,	L'élément A doit être présent et suivi de l'élément B	A,B
()	Les parenthèses permettent de regrouper des éléments afin de leur appliquer les autres opérateurs	(A,B)+

Exemple d'élément composé

- Exemple d'une DTD

```
<!ELEMENT personne (nom, prenom+, tel?, adresse) >
<!ELEMENT nom (#PCDATA) >
<!ELEMENT prenom (#PCDATA) >
<!ELEMENT tel(#PCDATA) >

<!ELEMENT adresse (ANY) >
```

- Un document XML conforme à la DTD

```
<personne>
  <nom>Hugo</nom>
  <prenom>Victor</prenom>
  <prenom>Charles</prenom>
  <tel>01120243</tel>
  <adresse><rue></rue><ville>Paris</ville></adresse>
</personne>
```

Déclaration d'attributs

<! ATTLIST balise Attribut Type Mode >

- Balise spécifie l'élément auquel est attaché l'attribut
- Attribut est le nom de l'attribut déclaré
- Type définit le type de donnée de l'attribut choisi parmi:
  - **CDATA**
    - Chaînes de caractères entre guillemets ("aa") non analysées
  - **Enumération**
    - Liste de valeurs séparées par |
    - *<! ATTLIST balise Attribut (Valeur1 | Valeur2 | …) >*
  - **ID et IDREF**
    - Clef et référence à clef
- Mode précise le caractère obligatoire ou non de l'attribut
  - *#REQUIRED, #IMPLIED (optionnel) ou #FIXED*

Exemple d'une DTD

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- DTD Livre : Livre.dtd -->
<!-- Caractérise un livre et son historique d'emprunts -->

<!ELEMENT Livre (Titre, Auteur, Couverture, Prix, ListeEmprunts)>
<!ELEMENT Titre (#PCDATA) >
<!ELEMENT Auteur (#PCDATA) >

<!ELEMENT Couverture EMPTY>
<!ATTLIST Couverture imgsrc CDATA #REQUIRED>

<!ELEMENT Prix (#PCDATA)>
<!ATTLIST Prix devise CDATA #REQUIRED>

<!ELEMENT ListeEmprunts (Emprunt*)>

<!ELEMENT Emprunt (Emprunteur,Sortie,Retour?)>
<!ELEMENT Emprunteur (#PCDATA)>
<!ELEMENT Sortie (#PCDATA)>
<!ELEMENT Retour (#PCDATA)>
```

DTD : Entité paramètre

- Permet la définition d'un groupe d'éléments sous un nom (macro)

<!ENTITY %nom "definition">

- Réutilisable dans une DTD par simple appel %nom;

- *Exemple :*

<!ENTITY %genres "(homme | femme)">

<!ATTLIST auteur genre %genres; #REQUIRED>

Quelques règles d'écriture

- **Modularité**
  - Définir dans des entités séparées les parties réutilisables
- **Précédence**
  - Regrouper les déclarations d'entités en tête
- **Abstraction**
  - Utiliser des entités pour les modèles de contenus
- **Spécificité**
  - Éviter les DTD trop générales
- **Simplicité**
  - Découper les DTD trop complexes

Insuffisance des DTD

- Pas de types de données à part du texte (#PCDATA)
- Expression de cardinalités limitée (‘?’, ‘*’ et ‘+’)
- Syntaxe spécifique (pas XML)
  - Difficile à interpréter
  - Difficile à traduire en schéma objets
- Propositions de compléments
  - XML-schema du W3C

XML Schéma

- Un schéma d'un document définit:
  - Les éléments possibles dans le document
  - Les attributs associés à ces éléments
  - La structure du document et les types de données
- Le schéma est spécifié en XML
  - Pas de nouveau langage
  - Balisage de déclaration
  - Utilise un espace de nom xs: (ou xsd:)
- Présente de nombreux avantages
  - Types de données personnalisés
  - Extensibilité par héritage et ouverture
  - Analysable par un parseur XML standard

XML Schéma: Objectifs

- Reprendre les acquis des DTD
  - Plus riche et complet que les DTD
- Permettre de typer les données
  - Eléments simples et complexes
  - Attributs simples
- Permettre de définir des contraintes
  - Existence obligatoire ou optionnelle
  - Domaines de valeurs, cardinalités, références
  - …
- S'intégrer à la nébuleuse XML
  - Espace de noms
  - Structure d'arbre logique

XML Schéma: Les types XML

- La base d'un schéma XML: l'élément

```
<xs:element   name="..."   type="..."/>
```

- Un élément peut avoir un type:
  - Simple si sa valeur a un type prédéfini en XML-SCHEMA (xs:string, xs:int, xs:decimal, xs:double…) ou une extension de ces types
  - Complexe s'il contient des sous éléments ou s'il comporte un attribut
    - ***xs:all*** tous les éléments doivent exister (peu importe l'ordre)
    - ***xs:choice*** un des éléments doit exister
    - ***xs:sequence*** tous les éléments doivent exister dans l'ordre spécifié

XML Schéma: Les types simples

- string
  Confirm this is electric
- byte
  -1, 126
- integer
  -126789, -1, 0, 1, 126789
- positiveInteger
  1, 126789
- negativeInteger
  -126789, -1
- hexBinary
  0FB7
- int
  -1, 126789675
- unsignedInt
  0, 1267896754
- boolean
  true, false 1, 0
- date
  1999-05-31
- anyURI
  http://www.example.com/e1.html#5
- language
  en-GB, en-US, fr
- dateTime
  1999-05-31T13:20:00.000-05:00
- Et beaucoup d'autres
  Short, long, float

XML Schéma: Les types complexes

- Définition d'objets complexes
  - <sequence> : collection ordonnée d'éléments typés
  - <all> : collection non ordonnée d'éléments typés
  - <choice>: choix entre éléments typés
- ***Exemple***:

```
<xs:complexType name="AddresseFR">
  <xs:sequence>
        <xs:element name="nom" type="xs:string"/>
        <xs:element name="rue" type="xs:string"/>
        <xs:element name="ville" type="xs:string"/>
        <xs:element name="codep" type="xs:decimal"/>
  </xs:sequence>
  <xs:attribute name="pays" type="xs:NMTOKEN" fixed="FR"/>
</xs:complexType>
```

XML Schéma: Les patterns

- Permet de définir des contraintes sur type simple prédéfini
- Appliquer des restrictions
  - *Exemple de restriction*

```
<xs:simpleType name="num5">
    <xs:restriction base="xs:string">
        <xs:pattern value="\d{5}"/>
    </xs:restriction>
</xs:simpleType>
```

- Autres facettes de restriction
  - *xs:enumeration, xs:length…*

XML Schéma: Réutilisation de types

- Type simple avec extension

```
<xs:simpleType name="num5">
    <xs:restriction base="xs:string">
        <xs:pattern value="\d{5}"/>
    </xs:restriction>
</xs:simpleType>
```

- Type complexe (séquence)

```
<xs:element name="livre">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Titre" type="xs:string"/>
            <xs:element name="Auteur" type="xs:string"/>
            <xs:element name="ISBN" type="num5"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

XML Schéma: Les occurrences

- Une bibliothèque contient au moins un livre

```
<xs:element name="biblio">
   <xs:complexType>
    <xs:sequence>
    <xs:element ref="livre" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>
</xs:element>
```

infini

XML Schéma: Les attributs

- Les éléments à contenu complexe avec attributs

```
<traduction langue="allemand" dateTraduction="2003-12-01">
        <traducteur>Michael</traducteur>
</traduction>
```

Les éléments en séquence

Les attributs

```
<xs:element name="traduction" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
          <xs:element name="traducteur" type="xs:string" minOccurs="1"
maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:attribute name="langue" use="required" type="xs:string"/>
     <xs:attribute name="dateTraduction" use="optional" type="xs:date"/>
    </xs:complexType>
</xs:element>
```

XML Schéma: Groupage d'éléments

```
<xs:group name="TitreAuteurISBN">
   <xs:sequence>
    <xs:element name="Titre" type="xs:string"/>
    <xs:element name="Auteur" type="xs:string"/>
    <xs:element name="ISBN" type="num5"/>
   </xs:sequence>
</xs:group>

<xs:element name="livre">
   <xs:complexType>
    <xs:sequence>
        <xs:group ref="TitreAuteurISBN"/>
         <xs:element ref="traduction"/>
    </xs:sequence>
   </xs:complexType>
</xs:element>
```

Bilan DTD et XML Schéma

- Les DTD définissent la grammaire des documents
- Les DTD sont écrites en SGML
- Elles sont de plus en plus souvent remplacées par des schémas.
- Le standard XML-Schéma est peu complexe

Passons aux Exercices!

XML: eXtensible Markup Language

Partie 3: La recherche dans un document XML

Xpath

Xpath: l'adressage dans XML

- **XPath est le langage qui permet d'adresser une partie ou plusieurs parties** d'un document, c'est à dire un ou plusieurs noeuds du document XML.
- Considérons cet exemple:

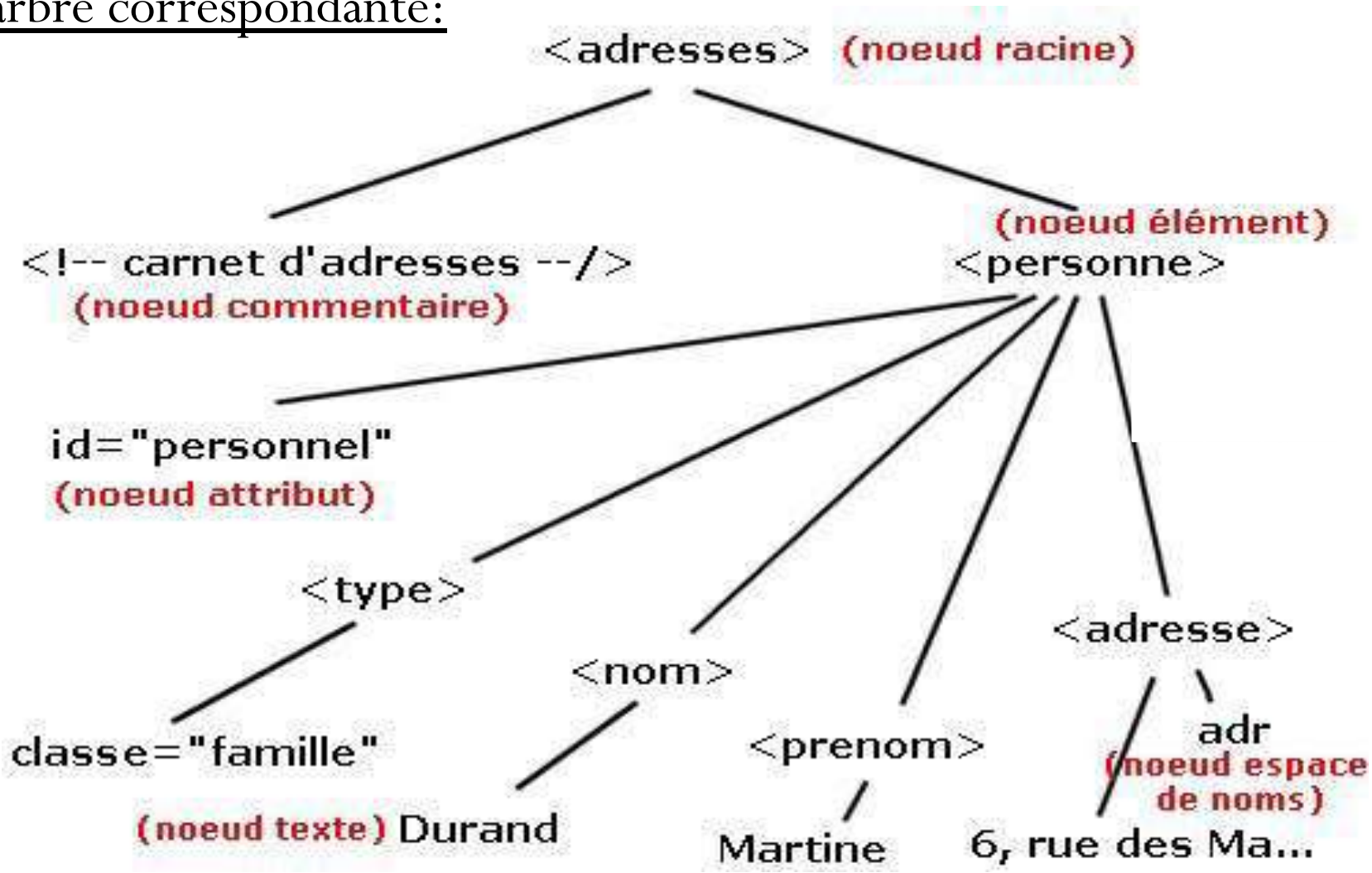
```
<adresses>
    <!--Carnet d'adresses-->
    <personne id= "personnel">
        <type classe= "famille"/>
        <nom>Durand</nom>
        <prenom>Martine</prenom>
        <adr:adresse>6, rue des Magnolias</adr:adresse>

    </personne>
</adresses>
```

- Première étape: Représenter l'arbre de ce document?

Xpath: l'adressage dans XML

L'arbre correspondante:



Xpath: l'adressage dans XML

- →La syntaxe de base **XPath** est semblable à l'adressage dans un système de fichiers.
- Syntaxe Xpath: */adresses/personne/nom*
  - *Résultat: <nom>Durand<nom>*
- *2ème exemple:*

```
<annuaire type="LDAP">
   <entree>
      <nom>Pierre Martin</nom>
      <phone>0556010101</phone>
   </entree>
   <entree>
      <nom>Martine Durand</nom>
   </entree>
</annuaire>
```

Xpath: l'adressage dans XML

Sym.	Rôle	Exemple	Résultat
/a/b	Liste des éléments dont le chemin dans l'arborescence correspond à la requête	/annuaire/entree/nom	<nom>Pierre Martin</nom> <nom>Martine Durand</nom>
//	On ne tient pas compte de la profondeur de l'élément dans l'arborescence	//nom	Idem
|	Composition de requêtes	//nom | //phone	<nom>Pierre Martin</nom> <phone>0556010101</phone> <nom>Martine Durand</nom>
*	Tous les éléments fils de la sélection	//entree/*	Idem

Xpath: l'adressage dans XML

1.Recherche par contexte

Sym.	Rôle	Exemple	Résultat
descendant	Fils d'un nœud et leurs descendants	/A/D/descendant::*	<G> <H/> </G>
ancestor	Père d'un nœud et ses ancêtres	/G/ancestor ::*	<A> <D id="12"> </D> </A>
child	Fils d'un nœud	/A/D/child::*	<G> <H/> </G>
attribute (ou@)	Attributs d'un noeud	//D/attribute::*	<D id= "10"> <D id="12">
parent (ou ..)	Parent d'un nœud	/A/D/G/H/parent::*	<G> <H/> </G>

```
<A>
   <B  att=''fr''>
      <C>Bonjour</C>
       <D/>
   </B>
   <B  att=''en''>
       <C>Hello</C>
    </B>
    <D  id='12'>
       <G>
          <H/>
        </G>
     </D>
     <EE/>
     <DE/>
</A>
```

Xpath: l'adressage dans XML

2. Recherche par prédicats :

Syno	Rôle	Exemple	Résultat
[i]	i^ème élément de la sélection	/A[4]	<D id="10"/>
[@att="val"]	Élément dont l'attribut att a pour valeur *val*	/A/B[@att='en']	<B att="en"> <C>Hello</C> </B>
text()	Tous les noeuds enfants de type textuels	//C[2]/text()	Hello
last()	Dernier élément de la sélection	/A[last()]	<DE/>
position()	Renvoi l'index de position du noeud relativement au noeud parent	//B[position()=2]	<B att="en"> <C>Hello</C> </B>

Xpath: l'adressage dans XML

2. Recherche par prédicats (suite) :

Sym.	Rôle	Exemple	Résultat
count( )	Compte les éléments de la sélection	//*[count(*)=2] (Liste des éléments ayant 2 fils)	<B att="fr"> <C>Bonjour</C> <D/> </B>
name( )	Nom de l'élément	//*[name()='G']	<G><H/></G>
contains( )	Condition sur les chaînes de caractère	//*[contains(name(),'E')]	<EE/> <DE/>
startswith( )	Condition sur les chaînes de caractère	//*[starts-with(name(),'D')]	<D id="12"> <G><H/></G> </D> <D id="10"/> <DE/>

Xpath: l'adressage dans XML

3.Utilisation d'opérateurs :

Type	Opérateurs
Booléen	and, or
Logique	!=, >=, >, <, <=
Opérations	+, -, *, div, mod

- **Requête:**

//*[@att='en' and name()='B']

- **Réponse:**

<B att="en">
<C>Hello</C>
</B>

Xpath: l'adressage dans XML

Quelques exemples supplémentaires:

Requête	Résultat
B[@att="en"]	sélectionne tous les enfants *B* du noeud contextuel qui ont un attribut *att* ayant la valeur *en*
D[@id="10"][1]	sélectionne le 1er enfant *D* du noeud contextuel qui ont un attribut *id* ayant la valeur *10*
D[1][@id="10"]	sélectionne le 1er enfant *D* du noeud contextuel si celui-là a un attribut *id* dont la valeur est *10*
//B[C="Hello"]	sélectionne tous les enfants *B* qui ont au moins un enfant *C* dont le contenu textuel est *Hello*
B[C]	sélectionne les enfants *B* du noeud contextuel qui ont au moins un enfant *C*
B[C and @att]	sélectionne tous les enfants *B* du noeud contextuel qui ont simultanément au moins un enfant *C* et un attribut *att*

Passons aux Exercices!

XML: eXtensible Markup Language

Partie 3: La recherche dans un document XML

Introduction à Xquery

D'où vient l'idée du Xquery?

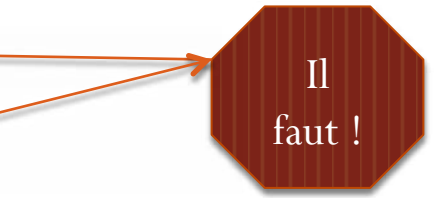
Le stockage d'informations??

La plupart des informations disponibles actuellement sont stockées ***dans des bases relationnelles.***

Le langage SQL est mature, et bien implanté!!

Peut-on adapter SQL aux données semi- structurées ?

- Récupération d'applications
- Récupération de compétences
- En quoi les données XML sont elles différentes ?



Xquery Vs SQL

	SQL	XML
Type d'arbre	Arbre à deux niveaux (tables et tuples)	Arbre très varié
Nature	Uniforme et répétitif (tous les comptes bancaires ont presque la même structure)	Très varié (chaque page est varié)
Les requêtes et les résultats	Retourne un ensemble de résultats homogènes	Les résultats d'une requêtes XML peuvent êtres de types différents et des structures complexes. Ces requêtes peuvent effectuer des transformations structurelles et retourne un document XML

Xquery Vs SQL

XML est très différent de SQL =>il faut un langage de requêtes dédié

Ok! Mais la tâche n'est pas simple!!!

Xquery : Les contraîntes du langage

- Respect du *modèle de données XML*
- Coordination avec *XML Schema*
- Doit supporter des *types de données simples et complexes*
- Doit supporter les *quantificateurs existentiels et universels*
- Doit supporter les opérations sur les *hiérarchies* et *séquences*
- Doit pouvoir *combiner* des informations de *plusieurs documents*
- Doit pouvoir transformer et *créer* des structures XML

…..

- Doit être en XML ? (XQuery)

Règles générales de Xquery

- XQuery est un langage sensible à la casse
- Les mots clés sont en minuscules
- Chaque expression a une valeur, et pas d'effet de bord
- Les expressions sont composables
- Les expressions peuvent générer des erreurs
- Les commentaires sont possibles

(: un commentaire :)

Xquery par les exemples

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>

<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
<book category="WEB">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
<author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author
  <year>2003</year>
  <price>49.99</price>
</book>

<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>

</bookstore>
```

Xquery par les exemples

On utilise des fonctions pour extarire des données à partir du fichier XML → Résultat de la requête: un document XML

1. **Doc(): fonction permet d'ouvrir un fichier XML**

```
doc("books.xml")
```

2. **Xquery utilise les expressions XPATH pour naviguer dans le document XML**

L'expression path:→ permet de sélectionner tous les éléments titres dans le fichier XML

```
doc("books.xml")/bookstore/book/title
```

Affichage

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

Xquery par les exemples

3 **Prédicat: Pour filtrer les informations sélectionnées**

```
doc("books.xml")/bookstore/book[price<30]
```

Affichage

```
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

Xquery par les exemples

4 **L'expression FLWOR: le parcours de tout le document**

FLWOR est l'acronyme de "For, Let, Where, Order by, Return".

L'expression XPATH

```
doc("books.xml")/bookstore/book[price>30]/title
```

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
return $x/title
```

L'expression XQUERY équivalente

Affichage du résultat

```
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

Xquery par les exemples

4 L'expression FLWOR: le parcours et le tri

For: sélectionne tous les livres sous bookstore element stockés dans la variable $x.

Where: selectionne uniquement les livres ayant un prix > 30.

order by pour trier selon l'élément title.

Return spécifie le résultat à retourner

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

On peut toujours utiliser les fonctions prédéfinis telles que: ***uppercase, substring***

Affichage du résultat

```
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

Xquery par les exemples

Xquery et HTML ☺

```
<ul>
{
for $x in doc("books.xml")/bookstore/book/title
order by $x
return <li>{$x}</li>
}
</ul>
```

Résultat

```
<ul>
<li><title lang="en">Everyday Italian</title></li>
<li><title lang="en">Harry Potter</title></li>
<li><title lang="en">Learning XML</title></li>
<li><title lang="en">XQuery Kick Start</title></li>
</ul>
```

Xquery par les exemples

Xquery et HTML ☺

Affiner l'affichage du résultat: → Afficher les nœuds textuels uniquement

```
<ul>
{
for $x in doc("books.xml")/bookstore/book/title
order by $x
return <li>{data($x)}</li>
}
</ul>
```

```
<ul>
<li>Everyday Italian</li>
<li>Harry Potter</li>
<li>Learning XML</li>
<li>XQuery Kick Start</li>
</ul>
```

Résultat

Xquery par les exemples

Les règles syntaxiques de Xquery ☺

- Xquery est sensible à la case
- Les éléments, les attributs et les variables doivent être valides.
- Les valeurs des chaînes de caractères doivent être en double quotes.
- Définition des variables => précédé par le caractère $ exemple: *$bookstore*
- XQuery commentaire → *(: and:) (: XQuery Comment :)*

Xquery par les exemples

Les expressions conditionnelles ☺

```
for $x in doc("books.xml")/bookstore/book
return if ($x/@category="CHILDREN")
then <child>{data($x/title)}</child>
else <adult>{data($x/title)}</adult>
```

```
<adult>Everyday Italian</adult>
<child>Harry Potter</child>
<adult>Learning XML</adult>
<adult>XQuery Kick Start</adult>
```

Résultat

Les expressions de comparaison:

=, !=, <,>,=<, >=, eq, ne, …….