XML

Un peu d’histoire

GML : première normalisation du balisage

- *Quoi* : unifier trois systèmes disparates
  - une application d'édition de texte,
  - une application de composition
  - une base pour l'interrogation documentaire.
- *Par qui* : Charles Goldfarb
- *Où* : IBM, New York
- *Pour qui* : un consortium d'avocats
- *Quand* : 1969

GML : comment

- Des documents contenant un mélange harmonieux :
  - d'information et
  - de méta-information de structure
- Documents compréhensibles par les machines
- Documents textuels faciles à mettre en œuvre
  - éditables à l'aide de n'importe quel éditeur de texte
- Comme les bases de données
  - On peut rechercher l'information par son contenu
  - On peut étiqueter l'information par « identificateur unique »

Un exemple de dialect GML (1978)

DCF (Document Composition Facility), un ancêtre de html

Le premier dialect GML à qui Goldfarb va donner une DTD.

```
:h1.Chapter 1:  Introduction
:p.GML supported hierarchical containers, such as
:ol.
:li.Ordered lists (like this one),
:li.Unordered lists, and
:li.Definition lists
:eol.
as well as simple structures.
:p.Markup minimization (later generalized and
  formalized in SGML),
allowed the end-tags to be omitted for the "h1" and
  "p" elements.
```

GML: un énorme succès

- La mise en œuvre est générique, elle s'applique à d'autres types de documents
  - Exemple : documenter un porte avion pour le DOD …
- 1986 norme ISO (ISO 8879) pour Standard GML (SGML)

Naissance d'HTML

- *Contexte* : 1991, généralisation des réseaux
- *Problème* : afficher en n'importe quel point d'un réseau de l'information conservée de manière répartie sur divers machines du réseaux
- *Qui* : Tim Berners-Lee, CERN

HTML versus XML

- HTML publication hypertexte sur écran
- XML généricité + échange de données
  - pas de balises normalisées, l'auteur crée ses propres balises
  - balisages plus strict pour simplifier le traitement des documents
  - la casse des caractères dans les noms d'éléments est significative
  - Structurer l'information selon sa propre logique

Les dates

1969 naissance de GML ----------

1980 premier draft sur SGML ----------

1986 norme ISO pour SGML

1991 naissance d'HTML et du premier navigateur ----

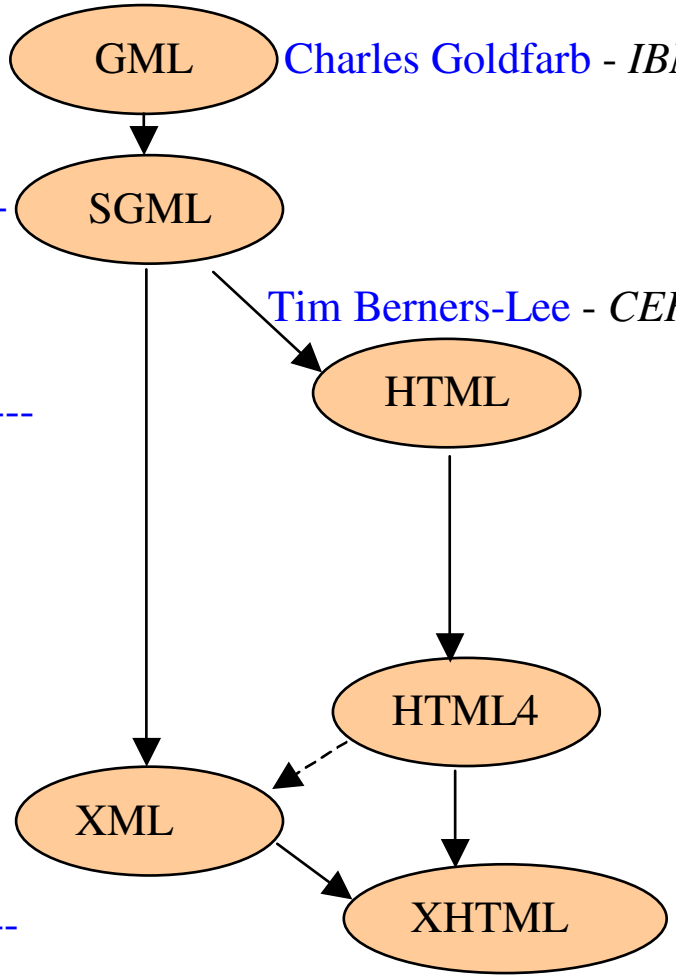
1993 explosion du WEB

1996 constat de limites d'HTML

1997 norme HTML 4 par le WWW ----------

1998 publication de "*XML, Java et le futur du WEB*" ----

1999 norme XHTML ----------



Rôle du document XML

Rôle du document XML

- Soit une entreprise, organisée en différents services
- Elle demande à un cabinet externe de réaliser des bilans de son activité.
- Ces bilans peuvent influencer le fonctionnement de plusieurs services, dont chacun possède ses particularités.
- Le cabinet fournit alors un document formalisé en vecteur d'information contenant ces bilans
- Ce document est ensuite traité par un logiciel qui établit un résultat personnalisé pour chaque service et propose également aux utilisateurs des fonctions de recherche et d'analyse

Rôle du document XML

- Un document XML
  - Sert de vecteur à l'information :
  - une manière universelle de représenter des données et leur sens dans un cadre précis

document XML : orienté document ou données ?

- Si les données sont élaborées par des êtres humains,
  - les fichiers XML produits sont orientés document.
    - Exemple: un livre, un article, un message…
- Si les données sont construites automatiquement par des programmes,
  - les fichiers XML sont orientés données.
    - Exemple: un sous-ensemble d'une base de données

La circulation XML : notion de bus

- Les données informatiques circulent aussi bien en interne, dans l'entreprise, que vers l'extérieur, auprès de services et de partenaires externes.
  - Importance du format de données
  - Différentes plateformes

- Le formalisme XML
  - neutralise les différences par un consensus de stockage,
  - garantit la neutralité des données transportées

Structure et validation d'un document XML

- On associe à un document XML un schéma,
  - peut être vu comme le schéma d'une base de données relationnelle.
- La validation d'un document XML garantit que la structure de données utilisée respecte ce schéma.
- La validation peut être considérée comme incontournable à certaines étapes de préparation du cadre d'exploitation.

Transformation et adaptation d'un document XML

- Un document XML peut être transformé
  - il n'est pas figé par un émetteur mais peut suivre, différentes étapes de modification.
- Le format XSLT (eXtensible Stylesheet Language Transformation) est un moyen pour adapter un document XML à un autre format XML.
- Exemple
  - une société dispose d'un ensemble de produits.
  - Produits présentés à la fois sur leur site Internet, dans un catalogue, et dans un logiciel interne pour les salariés…
  - Le formalisme XML tisse un lien entre ces différents médias, les données étant au cœur de l'activité,
  - la présentation n'étant plus qu'un processus de transformation

Les bases de données

XML et les bases relationnelles

- le formalisme XML peut-il remplacer les bases de données relationnelles telles que nous les connaissons ?
- Réponse: Non
  - il n'est optimisé ni en espace ni pour les manipulations que l'on peut opérer sur ce type de fichiers.
- Un document XML pourrait être davantage perçu comme une partie d'un système d'information,
  - il résout un problème de circulation de l'information à un moment donné.
- La recherche par SQL sera peut-être étendue pour ces types via la solution XQuery ; le standard SQL ISO travaille sur SQL/XML (http://www.sqlx.org/)

Les bases « natives » XML

- le formalisme XML peut utiliser des bases de données « native XML ».
  - relier des documents et pouvoir les manipuler plus facilement.
- deux formes de bases de données natives
  - celles gardant le texte du document XML tel quel et celles effectuant une conversion sous une forme objet
    - Exemple: DOM, qui est une standardisation objet d'un document XML
  - Celles pouvant s'appuyer sur des bases objets voire relationnelles
    - tables pour les éléments DOM : éléments, textes, commentaires…

Edition et manipulation d'un document XML

Cas des formats orientés document

- Édition avec un formulaire
  - Certaines solutions visent à analyser les schémas des fichiers XML pour générer un formulaire de saisie.
  - Exemples
    - EditLive! (http://www.ephox.com/)
    - InfoPath de Microsoft(http://office.microsoft.com/en-us/infopath/default.aspx)
- Éditeurs plus généralistes
  - s'adressent plutôt à des techniciens.
  - Il existe de nombreux produits, qui offrent tous la validation et la transformation
  - Ils se démarquent par certaines facilités
  - Exemples:
    - XMLSpy (http://www.altova.com/) payant
    - Stylus Studio (http://www.stylusstudio.com/) payant
    - EditiX (http://www.editix.com/) payant
    - XMLCooktop (http://www.xmlcooktop.com/) gratuit
    - XMLNotepad 2007 (http://msdn.microsoft.com/xml) un gratuit

Outils pour manipuler les documents XML

- Les parseurs XML
  - Rôle: analyser le document XML et servir de lien avec une application de traitement.
  - Il existe des parseurs
    - non validants qui n'offrent qu'une vérification syntaxique
    - validants qui offrent également le support des DTD/schéma W3C.
  - deux catégories de services
    - un service événementiel,
      - ne vise pas à représenter un document XML dans son intégralité
      - Exemple type SAX (Simple API for XML),exemple,
      - représentation du document n'est que partielle
    - un service objet, qui permet de représenter un document XML sous une forme objet,
      - exemple type DOM (Document Object Model)
      - représentation du document est complète.
    - Ces deux méthodes ont leurs avantages et inconvénients

Outils pour manipuler les documents XML

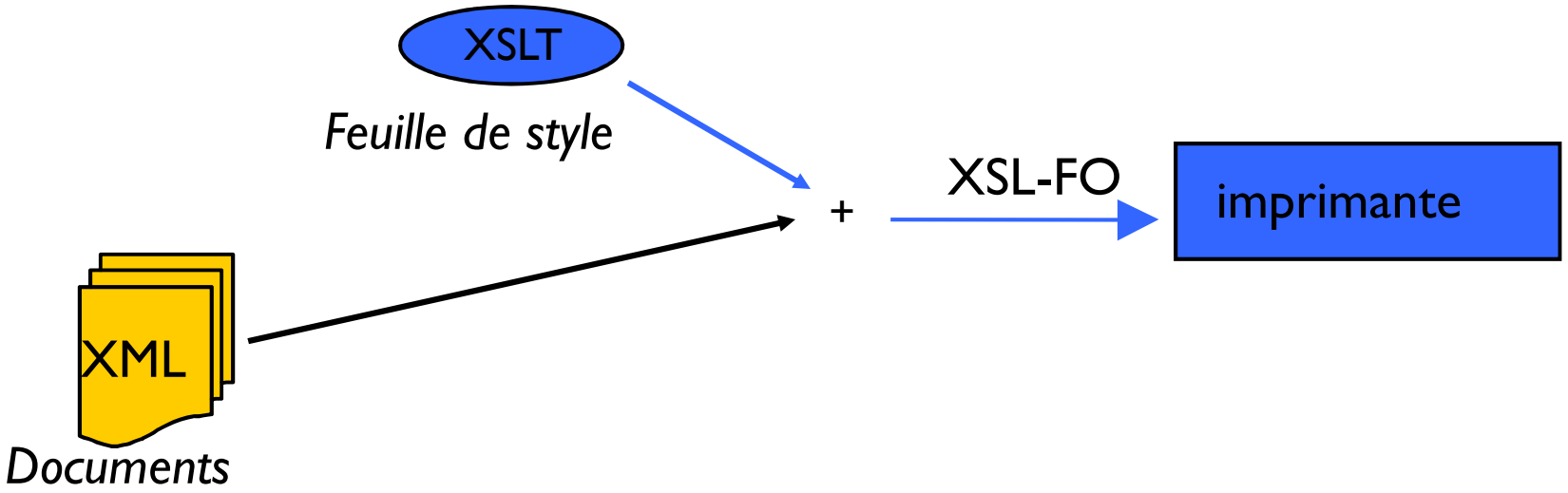
- Les parseurs XML
  - Microsoft XML Core Services (MSXML : http://msdn.microsoft.com)
    - une API composée d'un parseur validant, compatible SAX et DOM, et d'un moteur de transformation 1.0.
  - Xerces est disponible pour Java, C++ et Perl.
    - Open Source ,le plus abouti du marché, quelle que soit la plate-forme, en terme de respect du standard et de l'API (SAX,DOM).
  - Expat est un parseur réalisé en C (http://expat.sourceforge.net/),
    - dispose d'extensions pour SAX et DOM.
  - Piccolo (http://piccolo.sourceforge.net/).
    - un parseur non validant réalisé en Java (http://piccolo.sourceforge.net)
  - Un certain nombre de plates-formes, comme PHP et Java, disposent d'un parseur en standard.

Outils pour manipuler les documents XML

- Transformation d'un document XML
  - La transformation XSLT fonctionne en complément d'un parseur
  - API qui réalise le passage d'un document XML vers un document texte (souvent au format XML lui aussi)
  - La plupart des moteurs de transformation ne gèrent que la version XSLT 1.0.
    - Le toolkit MSXML de Microsoft (http://msdn.microsoft.com/) supporte la version 1.0.
    - Le groupe Apache gère le projet Xalan (http://xalan.apache.org/) pour Java et C++ avec support de la version 1.0.
    - Saxon est un projet Open Source avec également une licence commerciale. Il fonctionne pour Java et .NET et gère les versions 1.0 et 2.0.
    - Sablotron est une implémentation en C++ de la version 1.0 (http://www.gingerall.org/sablotron.html)

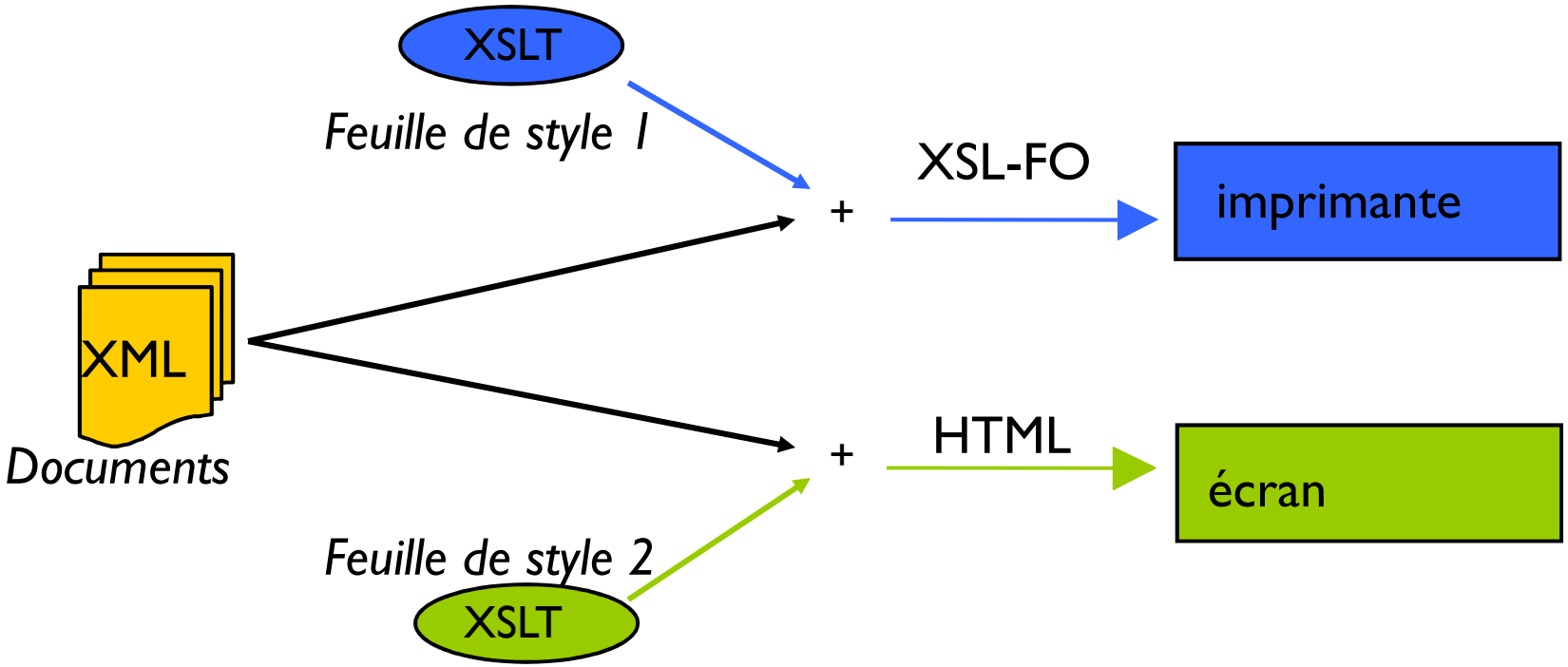
Outils pour manipuler les documents XML

- Transformation d'un document XML



Outils pour manipuler les documents XML

- Transformation d'un document XML



Outils pour manipuler les documents XML

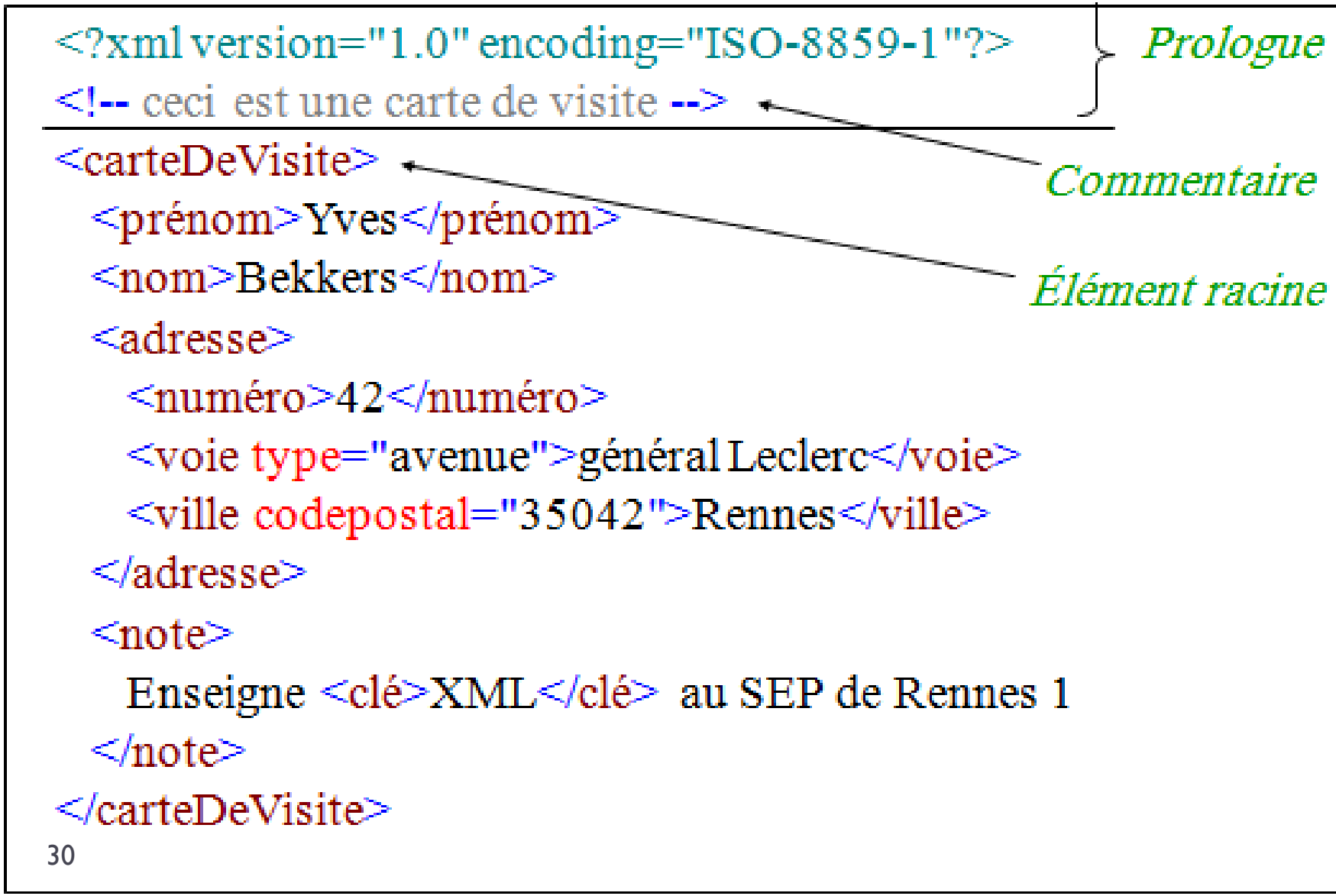
- Le format XSL-FO (Extensible Stylesheet Language Formatting Objects)
  - langage de présentation pour différents formats (PDF, RTF...).
  - Il y a peu d'outils capables de réaliser les transformations XSL-FO.
    - Ecrion (http://www.ecrion.com/)
      - gère en sortie les formats PDF et PostScript.
    - XEP de RenderX (http://www.renderx.com/)
      - gère en sortie les formats PDF et PostScript.
    - La seule solution Open Source est probablement FOP (Formatting Objects Processor) du groupe Apache (http://xmlgraphics.apache.org/fop/)
      - Elle gère en sortie les formats PDF, PostScript et RTF.

Outils pour manipuler les documents XML

- Le format SVG (Scalable Vector Graphics)
  - langage de description des dessins en 2D.
  - Il existe quelques plug-ins pour les navigateurs,
    - le plus connu étant SVG Viewer de adobe (http://www.adobe.com/svg/viewer/install/main.html)
  - le projet Batik est implémentation Open Source pour Java (http://xmlgraphics.apache.org/batikl) du groupe Apache

Structure des documents XML

Structure d'un document XML



L'en-tête : le prologue

- encapsulé par <? et ?>
- il n'existe pas d'espaces entre le début du document et cet élément
- Facultatif
- La version XML, soit 1.0 ou 1.1
- Le jeu de caractères employé (encoding)
  - Le parseur va avoir besoin de distinguer le rôle de chaque caractère,
    - certains étant réservés à la syntaxe
    - d'autres représentant des données.
  - XML s'appuie sur des standards ISO et Unicode (voir http://www.unicode.org/)
    - standard: ISO-8859-1.
      - Si l'encodage n'est pas précisé, c'est le standard UTF-8 qui est
    - Il existe beaucoup d'autres standards,
      - ISO-2022-JP, Shift_JIS, EUC-JP… UTF-16

L'en-tête : le prologue

- Le champ standalone désigne l'indépendance du document,
  - au sens où il n'existe aucun élément externe qui puisse altérer la forme finale du document XML fourni à l'application par le parseur (références d'entités, valeurs par défaut…)
  - Ce champ prend les valeurs yes ou no
  - Peut être ignoré

Les instructions de traitement

- n'ont pas de rôle lié aux données ou à la structuration de votre document
- servent à donner à l'application qui utilise le document XML des informations
  - Informations totalement libres et dépendent du concepteur de l'application de traitement
- Se positionne à n'importe quel endroit du document
- <?xml-stylesheet type="text/xsl" href="affichage.xsl"?>

La déclaration du type de document

- optionnelle sert à attacher une grammaire de type DTD (Document Type Definition) au document XML
- Elle est introduite avant la première balise (racine)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Date de création : 30/09/07 -->
<!DOCTYPE cours SYSTEM "cours.dtd">
<cours titre="XML">
    <intervenant nom="alexandre brillant">
    </intervenant>
    <plan>
        Introduction
        XML et la composition de documents
    </plan>
</cours>
```

La déclaration du type de document

- optionnelle sert à attacher une grammaire de type DTD (Document Type Definition) au document XML
- <!DOCTYPE racine SYSTEM "URI vers la DTD">
  - SYSTEM indique qu'il s'agit d'une DTD propre
    - L'alternative est le mot-clé PUBLIC
- La déclaration de type de document peut héberger un bloc d'instructions propre aux DTD
  - DTD interne
- **Exemple**

```
<!DOCTYPE cours [
<!ELEMENT cours ( intervenant, plan )>
<!ELEMENT intervenant EMPTY>
<!ELEMENT plan (#PCDATA)>
<!ATTLIST cours titre CDATA #REQUIRED>
<!ATTLIST intervenant nom CDATA #REQUIRED>
]>
```

Les nœuds élément

- Les éléments gèrent la structuration des données d'un document XML
- Pour décrire le contenu des éléments, on parle de modèle de contenu, on trouve :
  - Rien : l'élément est vide.
  - Du texte: valeur
  - Un ou plusieurs éléments fils,
    - élément parent à ne pas confondre avec un élément ancêtre

Les nœuds élément

- Question: identifier les éléments du document XML suivant

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<cours>
    <intervenant>
    Phileas
    </intervenant>
    <separateur/>
    <chapitre>
    Formation XML
        <para>Un paragraphe</para>
        <para>Autre paragraphe</para>
    </chapitre>
</cours>
```

Les nœuds élément

Exercice 1: Création d'un livre en XML

- On souhaite écrire un livre en utilisant le formalisme XML.
- Le livre est structuré
  - en sections (au moins 2),
  - en chapitres (au moins 2)
  - en paragraphes (au moins 2)
- Le livre doit contenir la liste des auteurs (avec nom et prénom).
- Tous les éléments doivent posséder un titre, sauf le paragraphe qui contient du texte.
- Proposez une structuration XML de ce document (avec 2 auteurs, 2 sections, 2 chapitres par section et 2 paragraphes par chapitre)
- Utiliser l'encodage ISO-8859-1
- Votre document sera nommé livre1.xml

Les attributs d'un élément

- Un attribut
  - un couple (clé, valeur) associé à la définition d'un élément
  - localisé dans la balise ouvrante de l'élément
  - peut avoir de 0 à n attributs uniques
  - complémentaire de l'élément
- Exemple
  - <auteur nom="brillant" prenom="alexandre">
    - …
  - </auteur>
  - <contact email='a@a.fr'/>

attributs ou éléments

- Ce qui s'écrit avec des attributs peut également l'être en s'appuyant uniquement sur des éléments
- Exemple :

```
<personne nom="brillant" prenom="alexandre"/>
```

```
<personne>
    <nom>
        brillant
    </nom>
    <prenom>
        alexandre
    </prenom>
</personne>
```

l'inverse n'est pas vrai car un attribut ne peut pas être répété dans un élément

attributs ou éléments

Exercice 2: Utilisation des attributs

- Conception de livre2.xml à partir de livre1.xml
- On souhaite compléter la structure du document livre1.xml par les attributs nom et prenom pour les auteurs et titre pour le livre, les sections et les chapitres.
- Votre document sera nommé livre2.xml
- Y a-t-il des simplifications possibles ?

Les nœuds textes

- Dans un document XML,
  - donnée ≅ texte associé à
    - l'attribut (sa valeur,)
    - ou à l'élément (son contenu)
  - Les données constituent le cœur du document, et tout le reste, le formalisme, ne sert qu'à séparer et classer ces données.
- Les données sont dites terminales dans l'arborescence XML,

Les nœuds textes

- Entités prédéfinies
- < équivalent de < (less than) ;
- > équivalent de > (greater than) ;
- & équivalent de & (ampersand) ;
- " équivalent de " (quote) ;
- ' équivalent de ' (apostrophe).

```
<calcul>
   if ( a<b et b>c) ...
</calcul>
```

devient

```
<calcul>
   If (a&lt;b et b&gt;c)
</calcul>
```

Les nœuds textes

- Entités prédéfinies
  - < équivalent de < (less than) ;
  - > équivalent de > (greater than) ;
  - & équivalent de & (ampersand) ;
  - " équivalent de " (quote) ;
  - ' équivalent de ' (apostrophe)
  - en trop grand nombre dans un même bloc peuvent alourdir inutilement le document
  - Dans le cas du contenu textuel d'un élément, nous disposons des sections CDATA (Character Data).
    - bloc de texte dont les caractères seront pris tel quel par le parseur

```
<![CDATA[
<element>C'est un document XML
</element>
]]>
```

Les entités du document

- Si on conçoit une DTD (Document Type Definition),
  - On peut à loisir créer nos propre entités et y faire référence dans nos documents
- La valeur de l'entité peut être interne ou externe (placée dans un autre fichier).
- Exemple

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE cours [
<!ENTITY auteur "Alexandre Brillant">
<!ELEMENT cours (#PCDATA)>
]>
<cours>
   Cours réalisé par &auteur;
</cours>
```

Les entités du document

Exercice 3: Utilisation des entités prédéfinies

- On se propose de créer un nouveau document livre2bis.xml reprenant livre2.xml.
- Placez dans 2 paragraphes un bloc de texte contenant l'extrait suivant :
  - <element id="10">></element>
  - Pour le premier paragraphe, employez les entités prédéfinies.
  - Pour le deuxième paragraphe, employez une section CDATA.

Quelques règles de syntaxe

- Le nom d'un élément ne peut commencer par un chiffre
  - Si le nom d'un élément est composé d'un seul caractère il doit être dans la plage [a-zA-Z], _, :, .
  - Avec au moins 2 caractères, le nom d'un élément peut contenir _, -, . et : plus les caractères alphanumériques
- Tous les éléments ouverts doivent être fermés
- Un élément parent est toujours fermé après la fermeture des éléments fils.
- Pour connaître les caractères autorisés,
  - grammaire XML du W3C disponible à l'adresse http://www.w3.org/TR/2006/REC-xml-20060816/#sec-well-formed

Quelques conventions de nommage

- Employer des minuscules pour les attributs et les éléments
- Éviter les accents dans les noms d'attributs et d'éléments pour des raisons de compatibilité avec les outils du marché qui proviennent souvent d'un univers anglo-saxon
- Délimiter les attributs des éléments par des guillemets
- Séparer les noms composés de plusieurs mots par les caractères -, _, . ou une majuscule

Les espaces de noms (namespace)

- groupes d'appartenance ou familles.
- Servent à délimiter la portée d'une balise, d'un attribut ou d'une valeur d'attribut,
  - évitent les ambiguïtés d'usage
- Espace de noms par défaut
  - précisé par un pseudo-attribut xmlns
  - ns désigne namespace).
  - valeur associée est une URL garantissant l'unicité de l'espace de noms
  - peu contrôlable sur un document de taille importante

```
<chapitre xmlns="http://www.masociete.com">
   <paragraphe>
   ...
   </paragraphe>
</chapitre>
```

```
<chapitre xmlns="http://www.masociete.com">
   <paragraphe xmlns="http://www.autresociete.com">
   ...
   </paragraphe>
</chapitre>
```

Les espaces de noms (namespace)

- L'espace de noms explicite
  - Notion de préfixe:
    - pseudo-attribut commençant par xmlns:prefixe.
    - n'a de sens que par rapport à l'URL associée
    - Une fois déclaré, il est employable uniquement dans l'élément le déclarant et dans son contenu.
    - L'emploi consiste à ajouter en tête de l'élément, de l'attribut ou d'une valeur d'attribut, le préfixe suivi de :.

```
<p:resultat xmlns:p="http://www.masociete.com">
</p:resultat>
```

```
Document 1 :
<p:res xmlns:p="http://www.masociete.com">
</p:res>
Document 2 :
<zz:res xmlns:zz="http://www.masociete.com">
</zz:res>
```

Les espaces de noms (namespace)

- L'espace de noms explicite
  - Notion de préfixe:
    - On peut déclarer et utiliser plusieurs espaces de noms grâce aux préfixes

```
<p:res xmlns:p="http://www.masociete.com" xmlns:p2="http://www.autresociete.com">
   <p2:res>
   </p2:res>
</p:res>
```

- La suppression d'un espace de noms
  - utiliser la valeur vide " "

```
<element xmlns="http://www.masociete.com">
   <autreelement xmlns="">
      .. Aucun d'espace de noms
   </autreelement>
   <encoreunelement>
      ... Espace de nom par défaut
   </encoreunelement>
</element>
```

Les espaces de noms (namespace)

Exercice 4: Utilisation des espaces de noms par défaut et avec préfixe

- Il s'agit de créer un document livre3.xml sur la base de livre1.xml en respectant les points suivants :
  - Mettez tous les éléments dans l'espace de noms http://www.masociete.com sans utiliser d'espace de noms par défaut.
  - Mettez la deuxième section dans un espace de noms http://www.monentreprise.com.
  - Mettez le dernier paragraphe du dernier chapitre de la dernière section sans espace de noms.

Les espaces de noms (namespace)

- Application d'un esspace de noms sur un attribut
  - via un préfixe sur un attribut ou une valeur d'attribut.

```
<livre xmlns:p="http://www.imprimeur.com" p:quantite="p:50lots">
   <papier type="p:A4"/>
</livre>
```

Les espaces de noms (namespace)

Exercice 5: Utilisation des espaces de noms sur des attributs

- Supposons que le livre des exercices précédents est maintenant disponible en plusieurs langues (au moins en français et en anglais).
- Proposez une méthode pour gérer tous les titres et paragraphes en plusieurs langues
- Créez un document livre4.xml à partir de livre1.xml

TP1: Génération des factures à base des fichiers XML

TP : Etude de cas

- Objectifs
  - Mettre en place un site web pour la facturation des produits inclus dans une commande
  - L'administrateur se connecte à l'interface web et saisie une commande. Chaque commande comporte plusieurs produits. Après validation, le système doit générer une facture sous format XML, qui doit comporter les produits achetés, et le total à payer
  - Les produits sont stockés dans une base de données SQL (id, Type, description, disponibilité, prix par quantité)
  - L'administrateur fait entrer dans le formulaire de saisie id du produit, et le nombre d'unités
  - Le formulaire doit permettre à l'administrateur de faire entrer le nombre de produit qu'il veut.

TP : Etude de cas

- Travail à faire
  - Proposer une modélisation du problème (use case et diagramme de classe)
  - Implémenter le système
- Note
  - le système doit vérifier la disponibilité du produit.
  - Utiliser programmation Orienté objet
  - Utiliser l'exemple suivant pour générer un fichier sous php

```php
$myFile = "testFile.txt";
$fh = fopen($myFile, 'w');
$myFile = "testFile.txt";
$fh = fopen($myFile, 'w') or die("can't open file");
$stringData = "chaine 1\n";
fwrite($fh, $stringData);
$stringData = "chaine 2\n";
fwrite($fh, $stringData);
fclose($fh);
```

Introduction aux définitions et aux DTD

Les nœuds éléments

- Les nœuds éléments `<!ELEMENT balise (contenu)>`
  - Cas d'une balise en contenant une autre

```
<personne>
    <nom>Mohammed Amine</nom>
</personne>
```

`<!ELEMENT personne (nom)>`

  - Élément comportant une valeur simple `<!ELEMENT nom (#PCDATA)>`
  - Cas d'une balise vide `<nom />` `<!ELEMENT nom EMPTY>`
  - Cas d'une balise vide ou pouvant contenir une valeur simple

```
<!-- valeur simple -->
<personne>
    <nom>Mohammed Amine</nom>
</personne>

<!-- vide -->
<personne>
    <nom />
</personne>
```

```
<!ELEMENT personne (nom)>
<!ELEMENT nom ANY>
```

Structurer le contenu des balises

- **Séquence**
  - permet de décrire l'enchaînement imposé des balises dont on sépare le nom par des **virgules**
    - <!ELEMENT balise (balise2, balise3, balise4, balise5, etc.)>
      - L'ordre des déclarions est important
- **Liste de choix**
  - permet de dire qu'une balise contient **l'une** des balises décrites séparées par une **barre verticale**.
    - <!ELEMENT balise (balise2 | balise3 | balise4 | balise5 | etc.)>
- **Balise optionnelle**
  - on fait suivre son nom par un **point d'interrogation**
    - <!ELEMENT balise (balise2, balise3?, balise4)>
- **Balise répétée optionnelle (0..*)**
  - on fait suivre son nom par une **étoile**
    - <!ELEMENT balise (balise2, balise3*, balise4)>
- **Balise répétée**
  - on fait suivre son nom par un **plus**
    - <!ELEMENT balise (balise2, balise3+, balise4)>

Les attributs

- **Syntaxe**
  - <!ATTLIST balise attribut type mode>
    - balise: nom de l'élément en question
    - **type**
      - liste des valeurs possibles
      - Texte non parsé (nombre, lettre, chaîne de caractères, etc.)
      - identifiant unique
      - référence à un identifiant unique
    - **mode**
      - attribut obligatoire: #REQUIRED
      - attribut optionnel : #IMPLIED
      - valeur par défaut
        - à écrire en dur dans la règle
      - attribut constant
        - obligatoire avec une valeur fixée
          - <!ATTLIST balise attribut type #FIXED value>

Les attributs

- **Syntaxe**
  - <!ATTLIST balise attribut type mode>
    - balise: nom de l'élément en question
    - **type**
      - liste des valeurs possibles
        - <!ATTLIST balise attribut (valeur 1 | valeur 2 | valeur 3 | etc.) mode>
        - Exemple

```
<!ATTLIST personne sexe (masculin|féminin) mode>
<!-- valide -->
<personne sexe="masculin" />

<!-- valide -->
<personne sexe="féminin" />

<!-- invalide -->
<personne sexe="autre" />
```

Les attributs

- **Syntaxe**
  - <!ATTLIST balise attribut type mode>
    - **type**
      - Texte non parsé (nombre, lettre, chaîne de caractères, etc.)
        - <!ATTLIST balise attribut CDATA mode>
        - Exemple:

```
<!ATTLIST personne sexe CDATA mode>
<!-- valide -->
<personne sexe="masculin" />

<!-- valide -->
<personne sexe="féminin" />

<!-- valide -->
<personne sexe="autre" />

<!-- valide -->
<personne sexe="12" />
```

Les attributs

- **Syntaxe**
  - <!ATTLIST balise attribut type mode>
    - **type**
      - identifiant unique
        - <!ATTLIST balise attribut ID mode>
        - Exemple

```
<!ATTLIST personne position ID mode>


<!-- valide -->
<personne position="POS-1" />
<personne position="POS-2" />
<personne position="POS-3" />

<!-- invalide -->
<personne position="POS-1" />
<personne position="POS-1" />
<personne position="POS-2" />
```

Les attributs

- **Syntaxe**
  - <!ATTLIST balise attribut type mode>
    - **type**
      - référence à un identifiant unique

        <!ATTLIST father id ID mode >

        <!ATTLIST child id ID mode father IDREF mode >

        Exemple

```
<!-- valide -->
<father id="PER-1" />
<child id="PER-2" father="PER-1" />

<!-- invalide -->
<!-- l'identifiant PER-0 n'apparaît nulle part -->
<father id="PER-1" />
<child id="PER-2" father="PER-0" />
```

Les attributs

- **Syntaxe**
  - <!ATTLIST balise attribut type mode>
    - **mode**
      - attribut obligatoire: #REQUIRED
      - Exemple

```
<!ATTLIST personne sexe (masculin|féminin) #REQUIRED>

<!-- valide -->
<personne sexe="masculin" />

<!-- valide -->
<personne sexe="féminin" />

<!-- invalide -->
<personne />
```

      - attribut optionnel : #IMPLIED
      - Exemple

```
<!ATTLIST personne sexe CDATA #IMPLIED>

<!-- valide -->
<personne sexe="masculin" />
<!-- valide -->
<personne sexe="féminin" />
<!-- valide -->
<personne sexe="15" />
<!-- valide -->
<personne />
```

Les attributs

- **Syntaxe**
  - <!ATTLIST balise attribut type mode>
    - **Mode**
      - valeur par défaut
        - à écrire en dur dans la règle
        - Exemple

```
<!ATTLIST personne sexe CDATA "masculin">

<!-- valide -->
<personne sexe="masculin" />

<!-- valide -->
<personne sexe="féminin" />

<!-- valide -->
<!-- l'attribut sexe vaut "masculin" -->
<personne />
```

Les attributs

- **Syntaxe**
  - <!ATTLIST balise attribut type mode>
    - **Mode**
      - attribut constant
        - obligatoire avec une valeur fixée
          - <!ATTLIST balise attribut type #FIXED value>
          - Exemple

```
<!ATTLIST objet devise CDATA #FIXED "Euro">

<!-- valide -->
<objet devise="Euro" />

<!-- invalide -->
<objet devise="Dollar" />

<!-- invalide -->
<objet />
```

Les entités

- alias permettant de réutiliser des informations au sein du document XML ou de la définition DTD
  - entités générales,
    - permettent d'associer un alias à une information afin de l'utiliser dans le document XML.
      - <!ENTITY nom "valeur">
  - entités paramètres,
    - n'apparaissent que dans les définitions DTD. Elles permettent d'associer un alias à une partie de la déclaration de la DTD
      - <!ENTITY % nom "valeur">
  - les entités externes

Les entités

- alias permettant de réutiliser des informations au sein du document XML ou de la définition DTD
  - entités générales,
    - permettent d'associer un alias à une information afin de l'utiliser dans le document XML.
      - <!ENTITY nom "valeur">
      - Exemple

```
<!ENTITY samsung "Samsung">
<!ENTITY apple "Apple">

<telephone>
    <marque>&samsung;</marque>
    <modele>Galaxy S3</modele>
</telephone>
<telephone>
    <marque>&apple;</marque>
    <modele>iPhone 4</modele>
</telephone>
```

Les entités

- alias permettant de réutiliser des informations au sein du document XML ou de la définition DTD
  - entités paramètres,
    - n'apparaissent que dans les définitions DTD. Elles permettent d'associer un alias à une partie de la déclaration de la DTD
      - <!ENTITY % nom "valeur">
      - Exemple

```
<!ATTLIST telephone marque (Samsung|Apple) #REQUIRED>

<!ENTITY % listeMarques "marque (Samsung|Apple) #REQUIRED">
<!ATTLIST telephone %listeMarques; >
```

Les entités

- alias permettant de réutiliser des informations au sein du document XML ou de la définition DTD
  - les entités externes
    - permettent d'associer un alias à une information afin de l'utiliser dans le document XML
      - <!ENTITY nom SYSTEM "URI">

```
<!ENTITY samsung SYSTEM "samsung.xml">
<!ENTITY apple SYSTEM "apple.xml">

<telephone>
    &samsung;
    <modele>Galaxy S3</modele>
</telephone>
<telephone>
    &apple;
    <modele>iPhone 4</modele>
</telephone>

<!-- Contenu du fichier samsung.xml -->
<marque>Samsung</marque>

<!-- Contenu du fichier apple.xml -->
<marque>Apple</marque>
```

DTD internes

- écrite dans le même fichier que le document XML, auquel elle lui est spécifique
  - dans **DOCTYPE**
    - On le place sous le prologue
    - Syntaxe : <!DOCTYPE racine [ ]>

Exemple: Une boutique possède plusieurs téléphones. Chaque téléphone est d'une certaine marque et d'un certain modèle représenté par une chaîne de caractère

```
<!DOCTYPE boutique [
  <!ELEMENT boutique (telephone*)>
  <!ELEMENT telephone (marque, modele)>
  <!ELEMENT marque (#PCDATA)>
  <!ELEMENT modele (#PCDATA)>
]>
```

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

<boutique>
  <telephone>
    <marque>Samsung</marque>
    <modele>Galaxy S3</modele>
  </telephone>

  <telephone>
    <marque>Apple</marque>
    <modele>iPhone 4</modele>
  </telephone>

  <telephone>
    <marque>Nokia</marque>
    <modele>Lumia 800</modele>
  </telephone>
</boutique>
```

DTD externes

- écrite dans un autre fichier que le document XML,
  - Extension: .dtd
  - il existe 2 types
    - DTD externes **PUBLIC**
      - généralement utilisées lorsque la DTD est une norme
      - Syntaxe: `<!DOCTYPE racine PUBLIC "identifiant" "url">`
      - Exemple:
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```
    - DTD externes **SYSTEM**
      - permet d'indiquer au document XML l'adresse (relative ou absolue) du document DTD
      - Syntaxe: `<!DOCTYPE racine SYSTEM "URI">`
      - Exemple: `<!DOCTYPE boutique SYSTEM "doc1.dtd">`
- Avec DTD externes, les doc XML ne sont plus autonomes,
  - Pour que le document contenant la DTD soit bien pris en compte,
    - Mettre la valeur de l'attribut standalone à "**no**"

DTD externes

- Exemple avec **EditiX**

```
<?xml version = "1.0" encoding="UTF-8" standalone="no" ?>
 <!DOCTYPE boutique SYSTEM "boutique.dtd">
 <boutique>
  <telephone>
    <marque>Samsung</marque>
    <modele>Galaxy S3</modele>
  </telephone>

  <telephone>
    <marque>Apple</marque>
    <modele>iPhone 4</modele>
  </telephone>

  <telephone>
    <marque>Nokia</marque>
    <modele>Lumia 800</modele>
  </telephone>
</boutique>
```

```
<!ELEMENT boutique (telephone*)>
<!ELEMENT telephone (marque, modele)>
<!ELEMENT marque (#PCDATA)>
<!ELEMENT modele (#PCDATA)>
```

Question

- créer une DTD à partir du document Sports.xml

Question

- Un examen se référence à un module ou un élément de module, dont il contient le code, et le titre. Il contient également une date, une durée, le nom de la filière, et éventuellement la section et le groupe de TD. La date comporte uniquement le mois et l'année.
- Ces éléments sont suivis d'une liste de questions de cours comporte 5 à 10 questions. Et d'une parties d'exercices. Chaque partie comporte un pou plusieurs exercices.
- Un exercice représente un mélange d'énoncé et de questions numérotées.
- On suppose qu'un examen comporte entre 3 à 5 exercices, dont chacun comporte 2 à 5 questions.
- Concevez une DTD externe et un arbre logique vous permettant de produire votre fichier xml

XML Schema Definition Language

Critiques des DTD

- Les DTD ne sont pas au format XML
  - nouveau langage avec sa propre syntaxe et ses propres règles.
    - pour exploiter une DTD, il faut utiliser, généralement,un outil différent de celui qui exploite un fichier XML.
- Les DTD ne permettent pas de typer des données.
  - impossible de préciser le type (entier, décimal, date, chaîne de caractères, etc.)

- **Schémas XML:** permet de définir un document XML
  - Fonctionnement analogique aux DTD

Apports des Schémas XML

- **Le typage des données**
  - Les **Schémas XML** permettent de *typer* les données.
  - Permettent de créer des types de données propres .
- **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**
  - Les **Schémas XML** s'écrivent grâce au langage XML.

Structure d'un schéma XML

- **L'extension du fichier: .xsd**
- la première ligne d'un Schéma XML est le prologue
  - <?xml version="1.0" encoding="UTF-8" ?>
- le **corps d'un Schéma XML** est constitué d'un ensemble de **balises**

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

  - <!-- Elément racine -->
  - <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  - </xsd:schema>

Référencer un schéma XML

- Le **référencement d'un schéma XML** se fait au niveau de l'élément racine du fichier XML grâce à l'utilisation de 2 attributs.
  - **L'espace de noms**
    - xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  - **La location**
    - xsi:schemaLocation="chemin_vers_fichier.xsd«
- Exemple

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

<racine xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="chemin_vers_fichier.xsd">

</racine>
```

LES ÉLÉMENTS SIMPLES

- Un **élément simple** est un élément qui ne contient qu'une valeur dont le type est dit **simple**.
  - Il ne contient pas d'autres éléments.
  - Exemple
    - balise qui ne contient aucun attribut et dans laquelle aucune autre balise n'est imbriquée.
    - un attribut d'une balise peut également être considéré comme un élément simple

```
<!-- Ne contient ni attribut ni aucun autre élément => élément simple -->
<nom>ROBERT</nom>

<!-- Contient un attribut => n'est pas un élément simple -->
<!-- Cependant l'attribut "sexe" est un élément simple -->
<personne sexe="masculin">Robert DUPONT</personne>

<!-- La balise personne contient d'autres éléments (les balises nom et prénom)
     => n'est pas un élément simple -->
<personne>
  <!-- Ne contient ni attribut ni aucun autre élément => élément simple -->
  <nom>DUPONT</nom>

  <!-- Ne contient ni attribut ni aucun autre élément => élément simple -->
  <prenom>Robert</prenom>
</personne>
```

LES ÉLÉMENTS SIMPLES

- Pour déclarer une balise comme un élément simple

```
<xsd:element name="mon_nom" type="xsd:mon_type" />
```

- Soient les balises suivantes

```
<nom>DUPONT</nom>
<prenom>Robert</prenom>
<age>38</age>
```

```
<xsd:element name="nom" type="xsd:string" />
<xsd:element name="prenom" type="xsd:string" />
<xsd:element name="age" type="xsd:int" />
```

- Valeur par défaut

```
<xsd:element name="mon_nom" type="xsd:mon_type" default="valeur_par_defaut"/>
```

- Valeur constante

```
<xsd:element name="prenom" type="xsd:string" fixed="Robert" />
```

LES ATTRIBUTS

- Déclaration

```
<xsd:attribut name="mon_nom" type="xsd:mon_type" />
```

  - Exemple

```
<personne sexe="masculin">Robert DUPONT</personne>
<xsd:attribut name="sexe" type="xsd:string" />
```

LES TYPES SIMPLES

- Les types chaînes de caractères
  - **Le type normalizedString**
    - basé sur le type **string** et représente une chaîne de caractères *normalisée* qui peut contenir tout à l'exception de tabulations, de sauts de ligne et de retours chariot.
  - **Le type token**
    - basé sur le type **normalizedString** et représente une chaîne de caractères normalisée sans espace au début ni à la fin.
  - **Le type language**
    - basé sur le type **token** et représente une langue identifiée par 2 lettres

```
<xsd:element name="langue" type="xsd:language" />

<langue>fr</langue>
<langue>en</langue>
<langue>en-GB</langue>
<langue>en-US</langue>
```

LES TYPES SIMPLES

- Les types chaînes de caractères
  - **Le type NMTOKEN**
    - basé sur le type **token** et représente une chaîne de caractères "simple", sans espace qui ne contient que les symboles suivants :
      - Des lettres.
      - Des chiffres.
      - Les caractères spéciaux **.-_** et **:**
    - Si la chaîne de caractères contient des espaces au début ou à la fin, ils seront automatiquement supprimés.
  - **Le type NMTOKENS**
    - représente une liste de **NMTOKEN** séparés par un espace

```
<xsd:attribut name="list" type="xsd:NMTOKENS" />

<balise list="A:1_B C-2.">contenu de la balise</balise>
<balise list="AZERTY 123456 QSDFGH">contenu de la balise</balise>
```

LES TYPES SIMPLES

- Les types chaînes de caractères
  - **Le type Name**
    - basé sur le type **token** et représente un **nom XML**, c'est-à-dire une chaîne de caractères sans espace qui ne contient que les symboles suivants :
      - Des lettres.
      - Des chiffres.
      - Les caractères spéciaux **.-_** et **:**
    - La différence avec le type **NMTOKEN** est qu'une chaîne de caractères de type **Name** doit obligatoirement commencer par une lettre, ou l'un des 2 caractères spéciaux suivants : _ et **:**
- **Le type NCName**
  - basé sur le type **Name**. Sauf qu'il ne peut pas contenir le caractère spécial **:**

LES TYPES SIMPLES

- Les types chaînes de caractères
  - **Le type ID**
    - basé sur le type **NCName**, et doit contenir des valeurs uniques. n'est utilisable qu'avec des attributs
  - **Le type IDREF**
    - fait référence à un **ID** existant dans le document XML..
  - **Le type IDREFS**
    - représente lui une liste de **IDREF** séparés par un espace.
    - il convient de n'utiliser le type **IDREFS** que pour un attribut.

```
<xsd:attribut name="enfants" type="xsd:IDREFS" />
<personne num="P1">Paul</personne>
<personne num="P2">Marie</personne>

<personne enfants="P1 P2">Jeanne</personne>
```

LES TYPES SIMPLES

- Les types chaînes de caractères
  - **Le type ENTITY**
    - permet de faire référence à une entité le plus souvent non XML et déclaré dans des fichiers DTD.
    - basé sur le type **NCName** et il convient de l'utiliser que pour un attribut.

```
<xsd:attribut name="marque" type="xsd:ENTITY" />
<!ENTITY samsung "Samsung">
<!ENTITY apple "Apple">

<telephone marque="apple">iPhone</personne>
<telephone marque="samsung">Galaxy SII</personne>
```

  - **Le type ENTITIES**
  - permet de faire référence à une liste d'**ENTITY** séparés par un espace. et ne doit être utilisé qu'avec un attribut

LES TYPES SIMPLES

- Les types chaînes de caractères

Type	Description	Commentaire
string	représente une chaîne de caractères	attention aux caractères spéciaux
normalizedString	représente une chaîne de caractères normalisée	basé sur le type string
token	représente une chaîne de caractères normalisée sans espace au début et à la fin	basé sur le type normalizedString
language	représente le code d'une langue	basé sur le type token
NMTOKEN	représente une chaîne de caractère "simple"	basé sur le type token applicable uniquement aux attributs
NMTOKENS	représente une liste de NMTOKEN	applicable uniquement aux attributs
Name	représente un nom XML	basé sur le type token
NCName	représente un nom XML sans le caractère :	basé sur le type Name

LES TYPES SIMPLES

- Les types chaînes de caractères

Type	Description	Commentaire
ID	représente un identifiant unique	basé sur le type NCName applicable uniquement aux ttributs
IDREF	référence à un identifiant	basé sur le type NCName applicable uniquement aux attributs
IDREFS	référence une liste d'identifiants	applicable uniquement aux attributs
ENTITY	représente une entité d'un document DTD	basé sur le type NCName applicable uniquement aux attributs
ENTITIES	représente une liste d'entités	applicable uniquement aux attributs

LES TYPES SIMPLES

- Les types date
  - **Le type duration**
    - représente une *durée* exprimée en nombre d'années, de mois, de jours, d'heures, de minutes et de secondes selon l'expression PnYnMnDTnHnMnS.
      - P : le début de l'expression.
      - nY : le nombre d'années (year) où n est un nombre entier.
      - nM : le nombre de mois (month) où n est un nombre entier.
      - nD : le nombre de jours (day) où n est un nombre entier.
      - T : sépare la partie date de l'expression de sa partie heure.
      - nH : le nombre d'heures (hour) où n est un nombre entier.
      - nM : le nombre de minutes (minute) où n est un nombre entier.
      - nS : le nombre de secondes (second) où n est un nombre entier ou décimal.

LES TYPES SIMPLES

- Les types date
  - **Le type duration**
    - représente une *durée* exprimée en nombre d'années, de mois, de jours, d'heures, de minutes et de secondes selon l'expression PnYnMnDTnHnMnS.

```
<xsd:element name="duree" type="xsd:duration" />

<!-- 42 ans et 6 minutes -->
<duree>P42YT6M</duree>

<!-- -2 heures -->
<duree>-PT2H</duree>

<!-- 2 jours -->
<duree>P2D</duree>

<!-- 10.5 secondes -->
<duree>PT10.5S</duree>
```

LES TYPES SIMPLES

- Les types date
  - **Le type date**
    - permet d'exprimer une date selon l'expression YYYY-MM-DD
      - YYYY : l'année (year) sur 4 chiffres ou plus.
      - MM : le mois (month) sur 2 chiffres.
      - DD : le jour (day) également sur 2 chiffres.

```
<xsd:element name="madate" type="xsd:date" />
<!-- 13 janvier 1924 -->
<madate>1924-01-13</madate>

<!-- 12 décembre 34 avant JC -->
<madate>-0034-12-12</madate>

<!-- 4 novembre 12405 -->
<madate>12405-11-04</madate>
```

LES TYPES SIMPLES

- Les types date
  - **Le type time**
  - permet d'exprimer une heure selon l'expression: hh:mm:ss.
    - hh : les heures (hour) sur 2 chiffres.
    - mm : les minutes (minute) sur 2 chiffres.
    - ss : les secondes (second) sur 2 chiffres entiers ou à virgule.

```
<xsd:element name="monheure" type="xsd:time" />
<!-- 10 heures et 24 minutes -->
<monheure>10:24:00</monheure>

<!-- 2,5 secondes -->
<monheure>00:00:02.5</monheure>
```

LES TYPES SIMPLES

- Les types date
  - **Le type dateTime**
    - Permet de représenter une date ET une heure selon l'expression YYYY-MM-DDThh:mm:ss.
    - T est une lettre de séparation
  - **Le type gYear**
    - représente une année sur 4 chiffres ou plus.
    - Dans le cas où l'on souhaite exprimer une année avant Jésus-Christ, un signe **-** peut-être placé devant l'expression.
  - **Le type gYearMonth**
    - représente une année et un mois selon l'expression :YYYY-MM

LES TYPES SIMPLES

- Les types date
  - **Le type gMonth**
    - représente un mois sur 2 chiffres précédés du symbole **--**

```
<xsd:element name="mois" type="xsd:gMonth" />
<!-- mars -->
<mois>--03</mois>

<!-- décembre -->
<mois>--12</mois>
```

  - **Le type gMonthDay**
    - représente un mois et un jour selon l'expression : --MM-DD.
  - **Le type gDay**
    - représente un jour sur 2 chiffres précédés du symbole **---**

LES TYPES SIMPLES

- Les types date

Type	Description
duration	représente une durée
date	représente une date
time	représente une heure
dateTime	représente une date et un temps
gYear	représente une année
gYearMonth	représente une année et un mois
gMonth	représente un mois
gMonthDay	représente un mois et un jour
gDay	représente un jour

LES TYPES SIMPLES

- Les types numériques
  - Float, double, decimal, integer, long, int, short, byte, nonPositiveInteger, negativeInteger, nonNegativeInteger, positiveInteger, unsignedLong, unsignedInt, unsignedShort, unsignedByte
- Les autres types

Type	Description
boolean	l'état vrai ou faux
QName	un nom qualifié
NOTATION	une notation
anyURI	une URI
base64Binary	une donnée binaire au format Base64
hexBinary	une donnée binaire au format hexadecimal

LES TYPES COMPLEXES

- Un **élément complexe** est un élément qui contient d'autres éléments ou des attributs.

```
<!-- la balise personne contient d'autres balises => élément complexe -->
<personne>
   <!-- la balise nom est un élément simple -->
   <nom>ROBERT</nom>
   <!-- la balise prenom est un élément simple -->
   <prenom>Axel</prenom>
</personne>

<!-- la balise personne possède un attribut => élément complexe -->
<personne sexe="feminin">Axel ROBERT</personne>
```

- Déclarer un élément complexe

```
<xsd:element name="mon_nom">
   <xsd:complexType>
      <!-- contenu ici -->
   <xsd:complexType>
</xsd:element>
```

LES TYPES COMPLEXES

- Les contenus des types complexes peuvent êtres : **Simples, « standards » , mixtes**

- On appelle **contenu simple**, le contenu d'un élément complexe qui n'est composé que d'attributs et d'un texte de type simple

```
<!-- contient un attribut et du texte -->
<prix devise="euros">35</prix>

<!-- contient un attribut et du texte -->
<voiture marque="Renault">Clio</voiture>
```

LES TYPES COMPLEXES

- **contenu simple**
  - Syntaxe XSD

```
<xsd:element name="mon_nom">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="mon_type">
        <xsd:attribute name="mon_nom" type="mon_type" />
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

  - Exemple

```
<prix devise="euros">35</prix>
<xsd:element name="prix">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:positiveInteger">
        <xsd:attribute name="devise" type="xsd:string" />
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

LES TYPES COMPLEXES

- **contenu standard**
  - contenu d'un élément complexe qui n'est composé que d'autres éléments (simples ou complexes) ou uniquement d'attributs.

```
<!-- contient d'autres éléments -->
<personne>
  <nom>DUPONT</nom>
  <prenom>Robert</prenom>
```

```
<!-- contient un attribut -->
<voiture marque="Renault" />
```

  - Déclaration d'une balise contenant un ou plusieurs **attributs**

```
<voiture marque="Renault" modele="Clio" />
<xsd:element name="voiture">
  <xsd:complexType>
    <xsd:attribut name="marque" type="xsd:string" />
    <xsd:attribut name="modele" type="xsd:string" />
  </xsd:complexType>
</xsd:element>
```

LES TYPES COMPLEXES

- **contenu standard**
  - Déclaration d'une balise contenant d'autres éléments

```
<xsd:element name="mon_nom">
   <xsd:complexType>
      <xsd:sequence>
         <!-- liste des éléments -->
      </xsd:sequence>
      <!-- listes des attributs -->
   </xsd:complexType>
</xsd:element>
```

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

```
<!-- valide -->
<personne sexe="masculin">
   <nom>DUPONT</nom>
   <prenom>Robert</prenom>
</personne>

<!-- invalide => les balises nom et prenom sont inversées -->
<personne sexe="masculin">
   <prenom>Robert</prenom>
   <nom>DUPONT</nom>
</personne>
```

LES TYPES COMPLEXES

- **contenu standard**
  - Type all: utilisé lorsque l'on veut spécifier que les éléments contenu dans un type complexe peuvent apparaître dans n'importe quel ordre. Ils doivent cependant tous apparaître une et une seule fois.

```
<xsd:element name="mon_nom">
   <xsd:complexType>
      <xsd:all>
         <!-- liste des éléments -->
      </xsd:all>
      <!-- listes des attributs -->
   </xsd:complexType>
</xsd:element>
```

```
<!-- valide -->
<personne sexe="masculin">
   <nom>DUPONT</nom>
   <prenom>Robert</prenom>
</prenom>

<!-- valide -->
<personne sexe="masculin">
   <prenom>Robert</prenom>
   <nom>DUPONT</nom>
</prenom>
```

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

LES TYPES COMPLEXES

- **contenu standard**
  - **Le choix:** utilisé lorsque l'on veut spécifier qu'un élément contenu dans un type complexe soit choisi dans une liste pré-définie.

```
<!-- valide -->
<personne sexe="masculin">
   <nom>DUPONT</nom>
</prenom>

<!-- valide -->
<personne sexe="masculin">
   <prenom>Robert</prenom>
</prenom>

<!-- invalide => les 2 balises prenom et nom
ne peuvent pas apparaître en même temps -->
<personne sexe="masculin">
   <prenom>Robert</prenom>
   <nom>DUPONT</nom>
</prenom>
```

```
<xsd:element name="mon_nom">
   <xsd:complexType >
      <xsd:choice>
         <!-- liste des éléments -->
      </xsd:choice>
      <!-- listes des attributs -->
   </xsd:complexType>
</xsd:element>
```

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

LES TYPES COMPLEXES

- **contenu standard**
  - Cas d'un type complexe encapsulant un type complexe

```xml
<?xml version="1.0" encoding="UTF-8"?>
<personne>
  <identite>
    <nom>NORRIS</nom>
    <prenom>Chuck</prenom>
  </identite>
</personne>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="personne">
    <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="identite">
      <xsd:complexType>
        <xsd:sequence>
        <xsd:element name="nom"  type="xsd:string"/>
        <xsd:element name="prenom"  type="xsd:string"/>
      </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

LES TYPES COMPLEXES

- **contenu mixte**
  - Cas d'un type complexe encapsulant un type complexe

```
<xsd:element name="mon_nom">
   <xsd:complexType  mixed="true">
      <!-- liste des éléments -->
   </xsd:complexType>
   <!-- liste des attributs -->
</xsd:element>
```

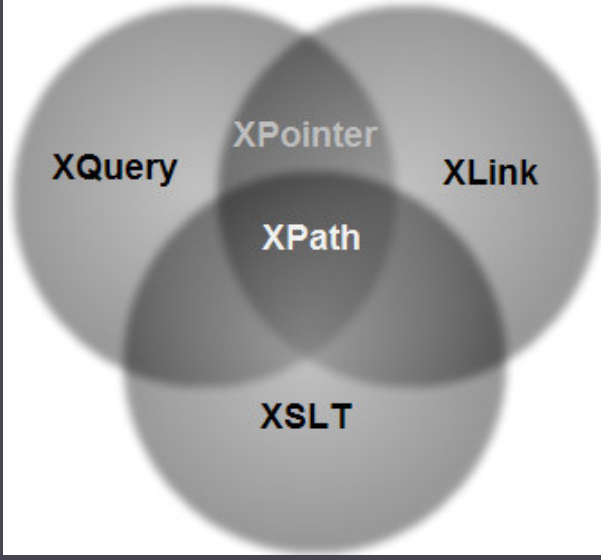
  - Exemple:

```
<facture><acheteur>Zozor</acheteur>, doit payer <somme>1000</somme>€.</facture>
```

```
<xsd:element name="facture">
   <xsd:complexType mixed="true">
      <xsd:sequence>
         <xsd:element name="acheteur" type="xsd:string" />
         <xsd:element name="somme" type="xsd:int" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

QUESTION

- Créer le Schéma XML d'un répertoire de personnes . Pour chaque personne, on souhaite connaître:
  - Son sexe (homme ou femme).
  - Son nom.
  - Son prénom.
  - Son adresse.
  - Un ou plusieurs numéros de téléphone (téléphone portable, fixe, bureau, etc.).
  - Une ou plusieurs adresses e-mail (adresse personnelle, professionnelle, etc.).



XPATH

Xpath?

- pas *vraiment* un langage
- juste une manière d'écrire pour accéder aux divers nœuds d'un arbre XML
- Arbre XML ≅ document XML,
- Le XPATH, comme le SQL, utilise la notion de requêtes

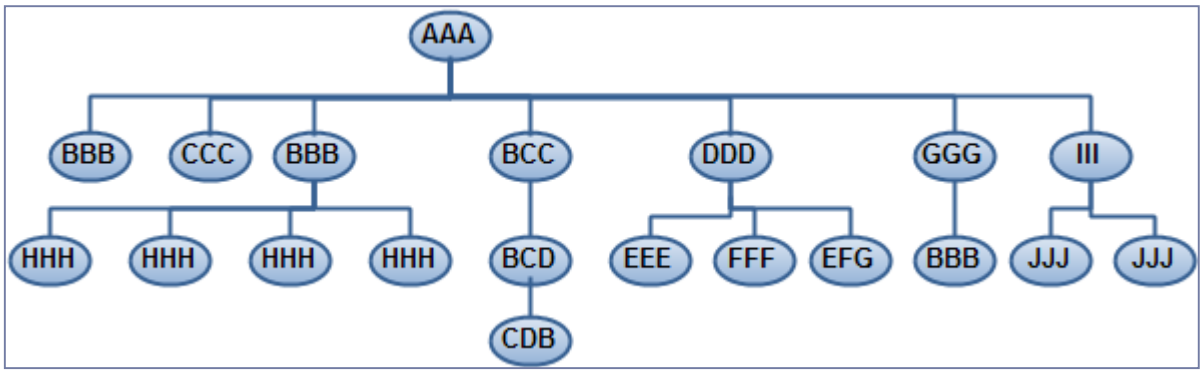
Xpath?

/nom_balise

correspond à la balise nommée « nom_Balise » qui se trouve être mère mais pas fille

//nom_Balise

toute balise de nom_Balise sera prise en compte : mère comme fille



(*) après un chemin désigne toutes les balises se trouvant après le début du chemin
cela fonctionne dans l'autre sens

(|), combine des chemins

(@) désigne un attribut d'une balise ou d'un document en général

Une expression suivie d'un **[n]** désigne l'énième élément d'une balise
/nom_balise[last()] sélectionne le dernier élément nommé nom_balise

Sélectionner des nœuds

Expression	Description
nomNœud	Sélectionne tous les nœuds ayant le nom "*nomNœud* »
/	Sélectionne depuis nœud racine
//	Sélectionne les nœuds dans le document à partir du nœud courant qui correspondent à la sélection, peu importe où ils se trouvent
.	Sélectionne le nœud courant
..	Sélectionne le parent du nœud courant
@	Sélectionne les attributs

Sélectionner des nœuds

- Question
  - À quoi correspondent les expressions suivantes
    - Bookstore
    - /bookstore
    - bookstore/bookSelects
    - //book
    - bookstore//book
    - //@lang

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

<bookstore>

<book>
  <title lang="eng">Harry Potter</title>
  <price>29.99</price>
</book>

<book>
  <title lang="eng">Learning XML</title>
  <price>39.95</price>
</book>

</bookstore>
```

Sélectionner des nœuds inconnus

Expression	Description
*	Correspond à n'importe quel nœud élément
@*	Correspond à n'importe quel attribut du nœud
node()	Correspond à n'importe quel nœud de n'importe quel type

- Question
  - À quoi correspondent les expressions suivantes
    - //bookstore/*
    - //*
    - //title[@*]

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

<bookstore>

<book>
  <title lang="eng">Harry Potter</title>
  <price>29.99</price>
</book>

<book>
  <title lang="eng">Learning XML</title>
  <price>39.95</price>
</book>

</bookstore>
```

Prédicats

- Ils sont utilisés pour trouver un nœud spécifique ou un nœud qui contient une valeur spécifique
- Ils sont toujours intégrés dans les crochets

Expression Path	Resultat
/bookstore/book[1]	Séléctionne le premier élément book qui est enfant de l'élément bookstore
/bookstore/book[last()]	Séléctionne le dernier élément book qui est enfant de l'élément bookstore
/bookstore/book[last()-1]	Séléctionne l'élément avant dernier de book qui est enfant de l'élément bookstore
/bookstore/book[position()<3]	Sélectionne les deux premiers éléments book qui sont enfants de l'élément bookstore
//title[@lang]	Sélectionne tous les titres des éléments ayant un attribut nommé lang
//title[@lang='eng']	Sélectionne tous les titres des éléments ayant un attribut nommé lang avec une valeur eng
/bookstore/book[price>35.00]	Sélectionne tous les titres des éléments de bookstore don't le pris est > 35.00
/bookstore/book[price>35.00]/title	Sélectionne tous les titres des éléments book du bookstore , ayant un prix > 35.00

Sélection de plusieurs chemins

- Pour sélectionner plusieurs chemins
  - opérateur | dans une expression XPath

- Question
  - À quoi correspondent les expressions suivantes
    - //book/title | //book/price
    - //title | //price
    - /bookstore/book/title | //price

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

<bookstore>

<book>
  <title lang="eng">Harry Potter</title>
  <price>29.99</price>
</book>

<book>
  <title lang="eng">Learning XML</title>
  <price>39.95</price>
</book>

</bookstore>
```

Xpath - fonctions

- count(nom_balise)
  - compte le nombre de balises qui ont l'enfant mis en paramètre, et le nombre d'enfant(s) mis après le symbole « égal »
    - Exemple
      [count(JJJ)=1], sélectionne toutes les balises ayant un seul et unique enfant JJJ
- name(nom_balise)
  - sélectionne toutes les balises de nom_balise et celles enfantes du chemin
    - Exemple:
    - [name()='GGG'], sélectionne toutes les balises GGG
- start-with(name(), 'B')
  - sélectionne toutes les balises dont le nom commence par un B
- contains(name(), 'G')]
  - sélectionne toutes les balises dont le nom contient un G

Xpath - fonctions

- string-length()
  - sélectionne des balises avec des nombres de caractères
  - Exemple
    string-length(name())=3
    - sélectionne toutes les balises dont le nom comporte 3 caractères
    - On peut utiliser (<) et (>)

Xpath - Axes

NomAxe	Resultat
ancestor	Sélectionne tous les ancêtres (parent, grand-parent, etc) du nœud courant
ancestor-or-self	Sélectionne tous les ancêtres (parent, grand-parent, etc) du nœud courant et le nœud courant lui-même
attribute	Séléctionne tous les attribut du nœud courant
child	Séléctionne tous les fils du nœud courant
descendant	Selects all descendants (children, grandchildren, etc.) du nœud courant
descendant-or-self	Sélectionne tous les descendants (fils, petits fils, etc.) du nœud courant et le nœud courant lui-même
following	Sélectionne tout dans le document après la balise de fermeture du nœud courant
following-sibling	Sélectionne tous les frères et sœurs après le nœud actuel
namespace	Sélectionne tous les nœuds d'espace de noms du nœud courant
parent	Sélectionne le parent du nœud courant
preceding	Sélectionne tous les nœuds qui apparaissent avant le nœud actuel dans le document, à l'exception des ancêtres, des attributs et des espaces de noms
preceding-sibling	Sélectionne tous les frères et sœurs avant le nœud courant
self	Sélectionne le nœud actuel

Xpath – Expression de chemin de localisation

- Chemin de localisation
  - Absolu
    - Commence par (/)
    - /step/step/...
  - Relatif
    - step/step/...
- Le chemin de localisation consiste en une ou plusieurs étapes, séparées par (/)
- Chaque étape est évaluée par rapport aux nœuds du document
- Une étape consiste à
  - un axe
    - l'arbre définit des relations entre les nœuds sélectionnés et le nœud présent
  - un nœud-test
    - identifie un nœud à l'intérieur d'un axe
  - zéro ou plusieurs prédicats (pour affiner l'ensemble de nœuds sélectionnés)
- La syntaxe d'une étape de localisation
  - 122 axisname::nodetest[predicate]

Xpath - Les « axes »

- axe « *descendant* »
  - sélectionne tous les descendants de la balise précédente du chemin.
  - /nom_balise/descendant::*
    - sélectionne tous les éléments descendants de nom_balise
  - /nom_balise/descendant::nom_balise2
    - sélectionne tous les éléments descendants de nom_balise, et nommés nom_balise2
- axe « *parent* »
  - sélectionne les parents directs de la balise précédente du chemin
  - //nom_balise/parent::*
    - sélectionne tous les parents directs de nom_balise
  - //nom_balise/parent::nom_balise2
    - sélectionne tous les parents de nom_balise nommés nom_balise2

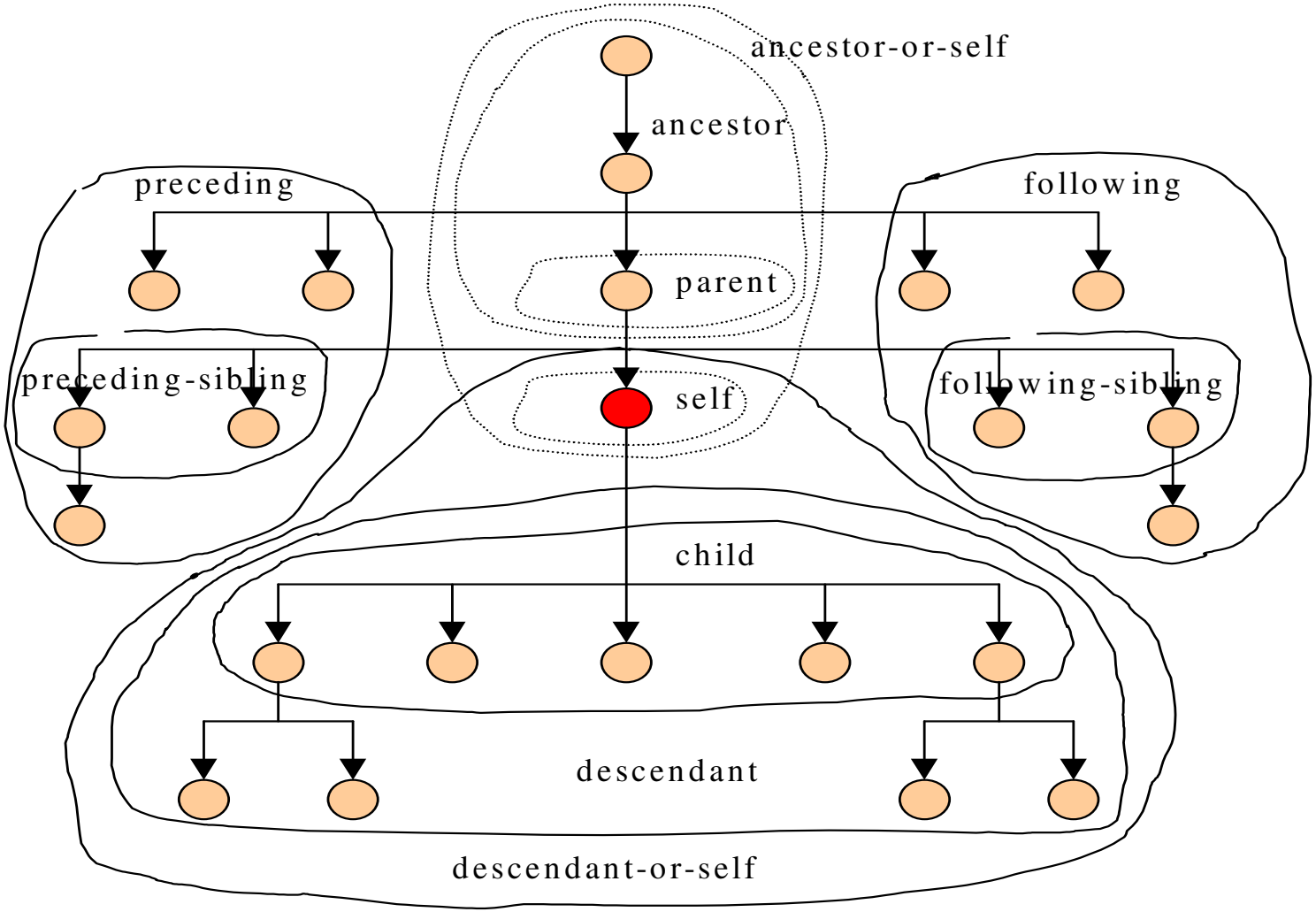
Xpath - « axes »

- axe « *ancestor* »
  - sélectionne tous les éléments ancêtres de la balise précédente dans le chemin
  - //nom_balise/ancestor::*
    - sélectionne toutes les balises ancêtres de nom_balise
- axe « *self* »
  - sélectionne tous les éléments du même nom que celui de la balise précédente dans le chemin
  - /nom_balise/nom_balise2/self::*,
    - Sélectionne tous les éléments nommés nom_balise2
- axe « *following* »
  - sélectionne tous les éléments qui suivent la balise précédente dans le chemin, sauf les éléments descendants de celle-ci.
  - /AAA/DDD/following::*
    - Selectionne tous les éléments situés après l'élément DDD, sauf les descendants de DDD

Xpath - « axes »

- axe « *preceding* »
  - sélectionne tous les éléments précédant la balise précédente dans le chemin, sauf les éléments descendants de celle-ci.
  - /AAA/DDD/preceding::*,
    - Sélectionne tous les éléments situés avant l'élément DDD , sauf les descendants de DDD.
- axes « *following-sibling* »
  - sélectionne tous les éléments frères qui suivent la balise précédente dans le chemin
  - /AAA/BCC/following-sibling::*
    - ne sélectionne que les balises DDD, GGG, et III.
- axe « *preceding-sibling* »
  - sélectionne tous les éléments frères précédant la balise précédente dans le chemin.
  - /AAA/BCC/preceding-sibling::*
    - ne sélectionne que les balises BBB, CCC et un autre élément BBB

Xpath - « axes »


ancestor-or-self
ancestor
preceding
following
parent
preceding-sibling
following-sibling
self
child
descendant
descendant-or-self

Xpath - « axes » - Exemple

Exemple	Resultat
child::book	Sélectionne tous les nœuds de book qui sont des enfants du nœud courant
attribute::lang	Sélectionne l'attribut lang du nœud courant
child::*	Sélectionne tous les éléments enfants du nœud courant
attribute::*	Sélectionne tous les attributs du nœud courant
child::text()	Sélectionne tous nœuds texte enfants du nœud courant
child::node()	Sélectionne tous les enfants du nœud courant
descendant::book	Sélectionne tous les éléments book descendants du nœud courant
ancestor::book	Sélectionne tous book ancestors of the current node
ancestor-or-self::book	Sélectionne tous les ancêtres du nœud courant, et le courant si c'est un nœud book
child::*/child::price	Sélectionne tous les prix des petits enfants du nœud courant

Exemple

exemples%20XML/Exemple_Xpath_Questionner_xml.html

```
<?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>
```

- Sélectionner tous les titres des nœuds

  /bookstore/book/title

- Sélectionner le titre du 1er livre sous la racine

  /bookstore/book[1]/title

- Sélectionner tous les prix

  /bookstore/book/price/text()

- Sélectionner les titres des nœuds ayant un prix >35

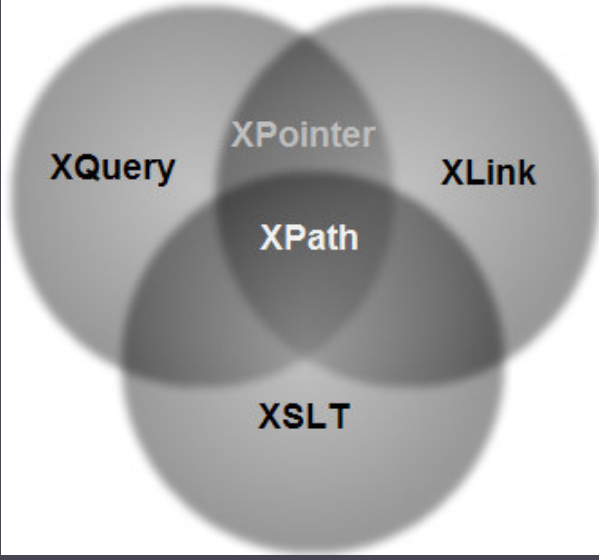
  /bookstore/book[price>35]/title

Xpath - « opérateurs»

Opérateur	Exemple
|	//book | //cd
+	6 + 4
-	6 - 4
*	6 * 4
div	8 div 4
=	price=9.80
!=	price!=9.80
<	price<9.80
<=	price<=9.80
>	price>9.80
>=	price>=9.80
or	price=9.80 or price=9.70
and	price>9.00 and price<9.90
mod	5 mod 2

Xpath

Exercices



Les bases de mise en forme avec XSLT

eXtensible Stylesheet Language Transformation

Définir une page XSLT

- Un fichier XSLT possède
  - Une extension : .xsl
  - la même version XML et le même encodage que la page XML
  - Si on utilise des entités dans la page, il vous faut les déclarer
    - quelques entités utiles

```
<!DOCTYPE xsl:stylesheet [
   <!ENTITY nbsp   " ">
   <!ENTITY copy   "&#169;">
   <!ENTITY reg    "&#174;">
   <!ENTITY trade  "&#8482;">
   <!ENTITY mdash  "&#8212;">
   <!ENTITY ldquo  "&#8220;">
   <!ENTITY rdquo  "&#8221;">
   <!ENTITY pound  "&#163;">
   <!ENTITY yen    "&#165;">
   <!ENTITY euro   "&#8364;">
]>
```

    - Les entités par défaut de XML n'ont pas besoin d'être déclarées
      - (« < », « & », « > », « " » et « ' »

Définir une page XSLT

- Pour ajouter une entité,
  - Il faut récupérer son nom et son code.
    - Se trouvant sur: http://www.w3.org/TR/REC-html40/sgml/entities.html
    - <!ENTITY le_nom "le_code " >
- Pour renseigner la version de XSLT et désigner le *namespace* (xmlns:xsl) des balises XSLT (par défaut *xsl*)

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

- Pour donner des informations sur le format de la page après transformation

```
<xsl:output method="html" encoding="utf-8" doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
                          doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"/>
```

- Pour indiquer quelles balises seront concernées

```
<xsl:template match="/">
```

Définir une page XSLT

- Exemple

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

<!-- New document created with EditiX at Tue Apr 30 13:08:52 WET 2013 -->

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

        <xsl:output method="html"/>

        <xsl:template match="/">
        <html>
                <body>

                <xsl:apply-templates/>
                </body>
        </html>
        </xsl:template>

</xsl:stylesheet>
```

Afficher du contenu XML

- Ensuite, on peut utiliser tout code qu'une page HTML peut gérer (balises HTML classiques, CSS et même JavaScript)
- Il faut refermer les balises *xsl* par la suite
  - </xsl:template>
  - </xsl:stylesheet>
- Utilisons la table ci-dessous pour y afficher le contenu des balises

```
<table width="1000" border="1" cellspacing="0" cellpadding="0">
    <tr>
        <th scope="col">ID</th>
        <th scope="col">Nom</th>
        <th scope="col">Type</th>
        <th scope="col">Prix</th>
    </tr>
        <tr>
        <td> </td>
        <td> </td>
        <td> </td>
        <td> </td>
    </tr>
</table>
```

Afficher du contenu XML

- Une balise XSL prend la forme
  - <xsl:nom_de_la_balise attributs="valeurs" />
- Pour afficher le contenu
  - <xsl:value-of select="path"/>
- Pour afficher toutes les informations de la première balise *jeu*

```
<td><xsl:value-of select="test/jeu/@id"/></td>
<td><xsl:value-of select="test/jeu/nom"/></td>
<td><xsl:value-of select="test/jeu/type"/></td>
<td><xsl:value-of select="test/jeu/prix"/></td>
```

```
<?xml version="1.0" encoding="utf-8"?>
<test>
    <jeu id="1">
        <nom>Guild Wars</nom>
        <type>Jeu de rôle en ligne</type>
        <prix>Environ 20 € l'épisode</prix>
    </jeu>
    <jeu id="2">
        <nom>Super Mario Galaxy</nom>
        <type>Jeu de plate-forme</type>
        <prix>30-50 €</prix>
    </jeu>
    <jeu id="3">
        <nom>Mario Sokoban</nom>
        <type>Jeu de réflexion</type>
        <prix>Gratuit</prix>
    </jeu>
</test>
```

ID	Nom	Type	Prix
1	Guild Wars	Jeu de rôle en ligne	Environ 20 € l'épisode

Lier la feuille XSLT au fichier XML

- Au début du fichier XML et après la première ligne, ajoutez la ligne suivante :
  - <?xml-stylesheet href="nom_de_la_feuille_xslt.xsl" type="text/xsl"?>
- Pour répéter l'affichages des contenus de balises, autant que ces dernières existent dans le document XML

```
<xsl:for-each select="test/jeu">
    <tr>
      <td><xsl:value-of select="@id"/></td>
      <td><xsl:value-of select="nom"/></td>
      <td><xsl:value-of select="type"/></td>
      <td><xsl:value-of select="prix"/></td>
    </tr>
</xsl:for-each>
```

ID	Nom	Type	Prix
1	Guild Wars	Jeu de rôle en ligne	Environ 20 € l'épisode
37	Super Mario Galaxy	Jeu de plate-forme	30-50 €
3	Mario Sokoban	Jeu de réflexion	Gratuit

Les filtres

- Restrictions à placer dans la balise select
- Se mettent entre [ ]
  - <xsl:for-each select="test/jeu[nom = 'Super Mario Galaxy']“>
- plusieurs opérateurs de comparaison peuvent s’appliquer

Égal	=
Non égal	!=
Supérieur	> (>)
Supérieur ou égal	>= (>=)
Inférieur	< (<)
Inférieur ou égal	<= (<=)

- Il est également possible de mettre plusieurs filtres en même temps
  - Utliser *or* et *and* entre les conditions.

Régions conditionnelles

- Pour afficher une zone que si une condition est remplie

```
<xsl:if test="a == b">
   <!--Le code-->
</xsl:if>
```

- ∃ une sorte de variante de *switch*

```
<xsl:choose>
   <xsl:when test="test = a">
      <!--Code-->
   </xsl:when>
   <xsl:when test="test = b">
      <!--Code-->
   </xsl:when>
   <xsl:otherwise>
   </xsl:otherwise>
         ...
</xsl:choose>
```

Commentaires

- La balise de base est
  - <xsl:comment>
  - Commentaire
  - </xsl:comment>

Les fonctions

- Pour déclarer une fonction, on utilise cette balise :

```
<xsl:template name="nomFonction">
    <!--Le code de votre fonction-->
</xsl:template>
```

- pour l'appeler
  - `<xsl:call-template name="nomFonction" />`
- déclarer un paramètre

```
<xsl:template name="nomFonction">
  <xsl:param name="parametre" select="0" />
</xsl:template>
```

- appeler une fonction avec un paramètre

```
<xsl:call-template name="NomFonction">
    <xsl:with-param name="parametre" select="375" />
</xsl:call-template>
```

- Ou alors

<xsl:with-param name="parametre">123</xsl:with-param>

Les fonctions

- Pour utiliser une variable dans une fonction,
  - Ajouter $ juste devant le nom de la variable
  - <xsl:template name="afficherVariale">
  - <xsl:param name="variable" />
  - <xsl:if test="$variable < 25">
  - <xsl:value-of select="$variable" />
    - <!--On a le droit de faire ça !-->
  - </xsl:if>
  - </xsl:template>

XML , XSLT et PHP

- Il est possible d'afficher une page XML mise en forme avec XSLT dans une page PHP
  - vérifier si l'extension XSL de PHP est activée
  - créer une page XML, une page XSLT (sans les balises de base d'un document HTML) et une page PHP
- Placer dans la page PHP, le code

```php
<?php
    $xslDoc = new DOMDocument();
    $xslDoc->load("sommaire.xsl");       //charger la feuille de style

    $xmlDoc = new DOMDocument();
    $xmlDoc->load("chapitre.xml");       //charger la page xml

    $proc = new XSLTProcessor();         // créer un nouveau container
                                            // qui doit contenir le résultat final
    $proc->importStylesheet($xslDoc);    // importer la feuille de style
    echo $proc->transformToXML($xmlDoc); //afficher le document xml
?>
```

Questions

- Soit le fichier Recette.xml
- Ecrivez une feuille XSLT qui produit un fichier HTML de titre « Recettes », dans lequel les recettes sont affichées les unes après les autres, en ajustant les la police et les espaces entres les titres des éléments