Chapitre I
Introduction a XML

Introduction

Le modèle relationnel n'est pas adapté aux données hétérogènes ou de structure mal définie.

Par exemple, si on veut stocker des informations clients de deux banques indépendantes qui représentent leurs clients de manière différente,

l’approche relationnelle semble peu adaptée à ce type de sauvegarde.

Introduction

Par contre, une représentation à l'aide du langage XML (eXtended Markup Language) permet de manipuler plus facilement ces informations irrégulières, grâce à la possibilité de structurer et d'annoter des informations sans la définition d'un schéma fortement structuré et contraignant.

Introduction

le langage XML représente les informations sous forme de documents textuels annotés et structurés par des balises dont la structure correspond à une arborescence d'éléments

Introduction

les données semi-structurées peuvent se voir comme une relaxation du modèle relationnel classique dans lequel on autorise une structure moins rigide et homogène des champs de données. Par conséquent, ce modèle de données est devenu très utile dans la représentation des documents de type multimédia, hypertexte, données scientifiques

I. Le langage XML

Le langage XML est un format textuel qui permet de créer des documents contenant des données semi-structurées.

il joue aujourd'hui un rôle de plus en plus important dans l'échange d'informations sur le Web.

L'utilisation la plus simple de XML consiste à créer des documents sans avoir défini de contrainte sur leur structure.

I. Le langage XML

Mais on peut également, comme dans le modèle relationnel, contraindre un (ensemble de) document(s) à respecter une structure définie sous forme de schémas ou types de documents

L'objectif de XML est de faciliter l'échange automatisé de contenus complexes (arbres, texte riche…) entre systèmes d'informations hétérogènes (interopérabilité).

I.1.Définition d’une DTD

afin d'imposer un peu plus de structure à un document XML, on lui associe une DTD (Document Type Definition).

Une DTD est un ensemble de règles qui spécifient pour chaque type d'élément les types de ses éléments fils, leur ordonnancement et leur fréquence.

I.1.Définition d’une DTD

Les DTDs définissent ainsi l'ensemble des balises qu'il est possible de trouver dans un document valide et établissent des contraintes sur l'ordre d'apparition de ces balises.

I.2. Validation d'un document XML

Un document est appelé "document XML" s'il est bien formé, ce qui signifie qu'il respecte les règles syntaxiques de XML.

Il est valide si, en plus, il respecte la grammaire du langage, contenue dans un fichier appelé DTD (Définition de Type de Document).

I.2.1.Validation syntaxique d'un document XML

Un document XML peut être découpé en 2 parties : le prologue et le corps.

Le prologue correspond à la première ligne de votre document XML. Il donne des informations

de traitement.

Exemple :

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

I.2.1.Validation syntaxique d'un document XML

Le corps d'un document XML est constitué de l'ensemble des balises qui décrivent les données.

Il y a cependant une règle très importante à respecter dans la constitution du corps :

une balise en paires unique doit contenir toutes les autres. Cette balise est appelée élément racine du corps.

I.2.1.Validation syntaxique d'un document XML

Un document XML est dit bien formé lorsque le document est correct sans toutefois posséder

une DTD.

Le prologue du document ne contient pas de Définition de Type de Document (DTD) et la structure arborescente du document respecte les standards XML (nom des balises et attributs, imbrication des marqueurs XML,...)

I.2.Validation d'un document XML par une DTD

Une DTD structure un document XML en définissant :

(1) le nom des éléments, leur contenu,

le nombre de fois et l'ordre d'apparition,

(2) les attributs éventuels et leurs valeurs par défaut et

(3) les noms des entités qui peuvent être utilisées.

I.2.Validation d'un document XML par une DTD

Les documents XML valides doivent respecter les règles données d'une DTD. La déclaration

d'une DTD doit apparaître après la déclaration XML, mais avant l’élément racine.

<!xml version="1.0" ....>

<!DOCTYPE élément_racine ....>

I.2.Validation d'un document XML par une DTD

La déclaration de la DTD peut contenir :

(1) la DTD elle-même à l'intérieur du fichier XML (DTD interne) ou

(2) une adresse URL qui indique le fichier contenant la DTD (DTD externe).

Le nom d'un élément utilisé dans le document XML doit être identique à celui déclaré dans la DTD.

A). Description des attributs d'une DTD

Un élément peut :

(1) contenir du texte,

(2) (2) contenir d'autres éléments,

(3) (3) contenir un mélange de texte et d'éléments (contenu mixte),

(4) être vide.

Chaque type d'élément doit être déclaré, cette déclaration respecte un des formats suivants :

(1) < ! ELEMENT NOM (CONTENU)>

(2) < ! ELEMENT NOM (CONTENU_MIXTE)*>

(3) < ! ELEMENT NOM ANY> n'importe quelles données

(4) < ! ELEMENT NOM EMPTY > élément vide (<NOM/>)

Exemple

- DTD

< ! ELEMENT personne (nom, prenom +, tel?, email, adresse >

< ! ELEMENT nom (#PCDATA) >

< ! ELEMENT prenom (#PCDATA) >

< ! ELEMENT tel (#PCDATA) >

< ! ELEMENT email (#PCDATA) >

< ! ELEMENT adresse (ANY) >

Exemple (Suite)

- Document XML associé

<personne>
<nom> Bennani </nom>
<prenom> Mohammed </prenom>
<prenom> Ali </prenom>
<tel> 0683000000 </tel>
<email> bennani@fsr.ac.ma </email>
<adresse> <rue> <rue/>
<ville>Rabat</ville></adresse>
</personne>

Description des attributs d’une DTD

La spécification du contenu d'un élément précise: le genre d'informations que l'élément peut contenir (texte, sous éléments, mixte) et les contraintes sur son contenu. Les mots clés de description du contenu sont :

Description des attributs d’une DTD

(#PCDATA) : Parsed Character Data, du contenu littéral.

(ELEMENT) : le sous-élément ELEMENT.

(ELEMENT1, ELEMENT2,...) : une liste d'éléments appelée séquence. L'ordre d'apparition des éléments doit être respecté dans le document XML.

(ELEMENT1|ELEMENT2|...) choix d'un sous-élément.

ELEMENT ? : zéro ou une fois.

ELEMENT+ : une ou plusieurs fois.

ELEMENT* : zéro ou plusieurs fois.

Description des attributs d’une DTD

Par exemple si on dit : qu'une liste de films lfilms contient des films, au moins un qu'un film contient un titre et zéro ou plusieurs acteurs (dans cet ordre), qu'un titre et un acteur sont des chaines de caractères PCDATA (Parsed Character Data) On écrira la DTD suivante :

< ! ELEMENT lfilms (film+)>

< ! ELEMENT film (titre, acteur*)>

< ! ELEMENT titre (#PCDATA)>

< ! ELEMENT acteur (#PCDATA)>

Exemple 1

<? xml version="1.0" encoding="iso-8859-1" standalone="yes"?>

< ! DOCTYPE parents [

< ! ELEMENT parents (fille, garcon)>

< ! ELEMENT fille (#PCDATA)>

< ! ELEMENT garcon (#PCDATA)>

]>

< parents >

< fille >Jalila</fille>

< garcon >Sami</garcon>

</ parents >

Exemple 2

```
<? xml version='1.0' encoding='ISO-8859-1'
standalone="yes"?>
< ! DOCTYPE BIBLIOTHEQUE [
< ! ELEMENT BIBLIOTHEQUE (LIVRE)* >
< ! ELEMENT LIVRE (AUTEUR, TITRE, EDITEUR)>
< ! ELEMENT AUTEUR (PRENOM, NOM) >
< ! ELEMENT TITRE (#PCDATA) >
< ! ELEMENT EDITEUR (NOM, ANNEE) >
< ! ELEMENT PRENOM (#PCDATA) >
< ! ELEMENT NOM (#PCDATA) >
< ! ELEMENT ANNEE (#PCDATA) >
]>
```

Exemple 2

```
<BIBLIOTHEQUE>
<LIVRE>
        <AUTEUR>
                <PRENOM>Rolf</PRENOM>
                <NOM> MAURERS</NOM>
        </AUTEUR>
        <TITRE>JAVA</TITRE>
        <EDITEUR>
                <NOM>Micro Application</NOM>
                <ANNEE> 1996</ANNEE>
        </EDITEUR>
</LIVRE>
</BIBLIOTHEQUE>
```

B) Description des attributs d’une DTD

La description des attributs se fait par une déclaration d'une liste d'attributs (ATTLIST)

La syntaxe est la suivante :

- FIXED signifie que l'attribut a une valeur fixe.

- REQUIRED signifie que l'attribut est obligatoire et n'a pas de valeur par défaut.

- IMPLIED signifie que l'attribut n'est pas obligatoire et n'a pas de valeur par défaut.

Exemple1

L'élément MESSAGE contient des données textuelles et peut contenir un attribut nommé LANGUE, sa valeur par défaut est "Français".

< ! ELEMENT MESSAGE (#PCDATA)>

< ! ATTLIST MESSAGE LANGUE CDATA "Français">

Exemple1

On peut, dans une même déclaration ATTLIST, définir plusieurs attributs associés au même élément :

< ! ATTLIST IMG WIDTH CDATA "100" HEIGHT CDATA "100">

Exemple1

Nous pouvons limiter la liste de valeurs possibles pour un attribut. On le définit comme un type énuméré. Nous déclarons un attribut format d'un élément img.

Cet attribut peut prendre une valeur parmi GIF, JPEG et PNG.La valeur par défaut est GIF.

< ! ELEMENT img EMPTY>

<! ATTLIST img format (GIF|JPEG|PNG) "GIF">

Exemple2

<! ELEMENT personne (nom, prenom+, tel?, email,adresse >
<!ELEMENT nom (#PCDATA) >
<!ELEMENT prenom (#PCDATA) >
<!ELEMENT tel (#PCDATA) >
<!ELEMENT email (#PCDATA) >
<!ELEMENT adresse ANY>
<! ATTLIST personne
age CDATA #IMPLIED
genre (Masculin | Feminin) #REQUIRED >
<!ELEMENT auteur (#PCDATA) >
<!ATTLIST auteur
genre (Masculin | Feminin) #REQUIRED
ville CDATA #IMPLIED>
<!ELEMENT editeur (#PCDATA) >
<!ATTLIST editeur
ville CDATA #FIXED "Rabat">

C) Attributs ID et IDREF

Ce type sert à indiquer que l'attribut concerné peut servir d'identifiant dans un fichier XML.

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE Document [
<!ELEMENT Document (Personne*)>
<!ELEMENT Personne (#PCDATA)>
<!ATTLIST Personne PNum ID #REQUIRED>
<!ATTLIST Personne Mere IDREF #IMPLIED>
<!ATTLIST Personne Pere IDREF #IMPLIED>
]>
```

< Document >

< Personne PNum = "P1">Latifa<//Personne>

< Personne PNum = "P2">Rachid</Personne>

< Personne PNum = "P3" Mere ="P1" Pere ="P2">Ali</Personne>

< Personne PNum = "P4" Mere ="P1" Pere ="P2">Samia</Personne>

</Document >

Les espaces de noms

- Un problème apparaît si on mélange deux textes XML dont les éléments ont le même nom.
- Pour régler ce problème on enrichit le nom de l'élément par l'identification de la source de données (URI) dans laquelle l'élément a défini.
- Le document importateur doit utiliser cette identification qui préfixe les éléments importés
- Les espaces de noms doivent être utilisés si le document XML rédigé est destiné à être mélange à d'autres sources

Les espaces de noms

Exemple :

<produit> <nom>...</nom> <desc>...</desc> </produit>	<dil:produit xmlns:i3s='http://www.i3s.unice.fr'> <i3s:nom>...</i3s:nom> <i3s:desc>...</i3s:desc> </i3s:produit>

On peut fixer l'espace de noms par défaut avec la syntaxe :
<produit xmlns='http://www.i3s.unice.fr'>
<nom>...</nom> <desc>...</desc>
</produit>

Insuffisance des DTD

- Pas de types de données
  - difficile à interpréter par le récepteur
  - difficile à traduire en schéma objets
- Pas en XML
  - langage spécifique

XML schéma

- Un schéma d'un document définit:
  - les éléments possibles dans le document
  - les attributs associés à ces éléments
  - la structure du document
  - les types de données
- Le schéma est spécifié en XML
  - pas de nouveau langage
  - balisage de déclaration
  - espace de nom spécifique xsd: ou xs:
- Présente de nombreux avantages
  - structures de données avec types de données
  - extensibilité par héritage et ouverture
  - analysable à partir d'un parseur XML standard

Objectifs des schémas

- 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, optionnel
  - Domaines, cardinalités, références
  - Patterns, ...
- S'intégrer à la galaxie XML
  - Espace de noms
  - Infoset (structure d'arbre logique)

Le modèle des schémas

- Déclaration des éléments et attributs
  - Nom
  - Typage similaire à l'objet
- Spécification de types simples
  - Grande variété de types
- Génération de types complexes
  - Séquence (Sequence)
  - Choix (Choice)
  - Tas (All)

Les types simples (1)

- string
  - Confirm this is electric
- normalizedString
  - Confirm this is electric
- token
  - Confirm this is electric
- byte
  - -1, 126
- unsignedByte
  - 0, 126
- base64Binary
  - GpM7
- hexBinary
  - 0FB7
- integer
  - -126789, -1, 0, 1, 126789
- positiveInteger
  - 1, 126789
- negativeInteger
  - -126789, -1
- nonNegativeInteger
  - 0, 1, 126789
- nonPositiveInteger
  - -126789, -1, 0
- int
  - -1, 126789675
- unsignedInt
  - 0, 1267896754

Les types simples (2)

- long
  - -1, 12678967543233
- unsignedLong
  - 0, 12678967543233
- short
  - -1, 12678
- unsignedShort
  - 0, 12678
- decimal
  - -1.23, 0, 123.4, 1000.00
- float
  - -INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
- double
  - -INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
- boolean
  - true, false 1, 0
- time
  - 13:20:00.000, 13:20:00.000-05:00
- dateTime
  - 1999-05-31T13:20:00.000-05:00
- duration
  - P1Y2M3DT10H30M12.3S
- date
  - 1999-05-31
- gMonth
  - --05--
- gYear
  - 1999

Les types simples (3)

- gYearMonth
  - 1999-02
- gDay
  - ---31
- gMonthDay
  - --05-31
- Name
  - shipTo
- QName
  - po:USAddress
- NCName
  - USAddress
- anyURI
  - http://www.example.com/,
  - http://www.example.com/doc.html#ID5
- language
  - en-GB, en-US, fr
- ID
  - "A212"
- IDREF
  - "A212"
- IDREFS
  - "A212" "B213"
- ENTITY
- ENTITIES
- NOTATION
- NMTOKEN, NMTOKENS
  - US
  - Brésil Canada Mexique

Commandes de base xsd:

- element : association d'un type à une balise
  - attributs name, type, ref, minOccurs, maxOccurs, ...
- attribute : association d'un type à un attribut
  - attributs name, type
- type simple : les multiples types de base
  - entier, réel, string, time, date, ID, IDREF, …,
  - extensibles par des contraintes
- type complexe : une composition de types
  - définit une agrégation d'éléments typés

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

```
<xsd:complexType name="Addresse">
   <xsd:sequence>
         <xsd:element name="nom" type="xsd:string"/>
         <xsd:element name="rue" type="xsd:string"/>
         <xsd:element name="ville" type="xsd:string"/>
         <xsd:element name="codep" type="xsd:decimal"/>
    </xsd:sequence>
   <xsd:attribute name="pays" type="xsd:NMTOKEN" />
</xsd:complexType>
```

Héritage de types

- Définition de sous-types par héritage
  - Par extension : ajout d'informations
  - Par restriction : ajout de contraintes
- Possibilité de contraindre la dérivation

- Exemple :

```
<xsd:complexType name="AdresseC">
  <xsd:complexContent>
    <xsd:extension base="Adresse">
      <xsd:sequence>
        <element name="continent"
          type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Les patterns

- Contraintes sur type simple prédéfini
- Utilisation d'expression régulières
  - Similaires à celles de Perl
- Exemple

```
<xsd:simpleType name="NumItem">
  <xsd:restriction base="xsd:string">
     <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
</xsd:simpleType>
```

XML Schema : exemple (1)

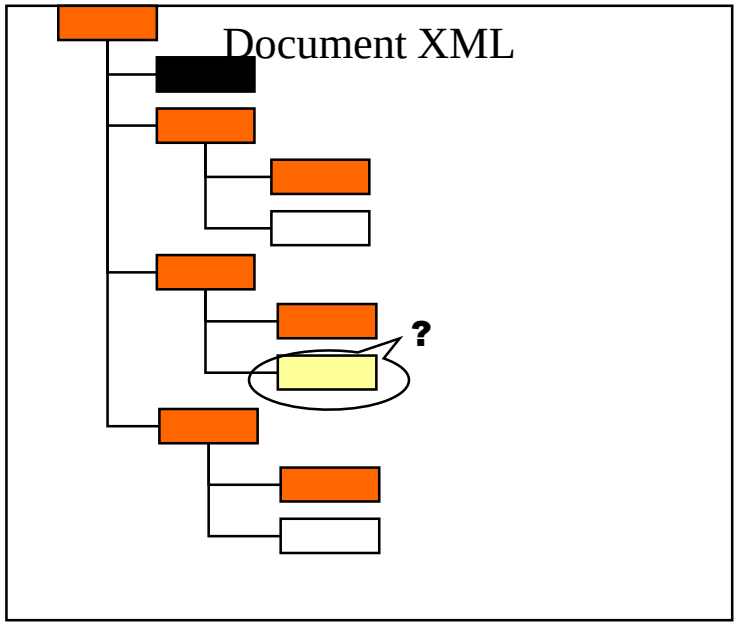
```
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">

<xsd:element name="commande" type="CommandeType"/>
<xsd:element name="commentaire" type="xsd:string"/>
<xsd:complexType name="CommandeType">
    <xsd:sequence>
        <xsd:element name="livrer" type="Adresse"/>
        <xsd:element name="facturer" type="Adresse"/>
        <xsd:element ref="commentaire" minOccurs="0"/>
        <xsd:element name="produits" type="ProduitType"/>
    </xsd:sequence>
    <xsd:attribute name="date_com" type="xsd:date"/>
</xsd:complexType>
```

XML Schema : exemple (2)

```
<xsd:complexType name="ProduitType">
    <xsd:sequence>
        <xsd:element name="produit" minOccurs="0" maxOccurs="unbounded">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="nom_prod" type="xsd:string"/>
                    <xsd:element name="quantite">
                      <xsd:simpleType> <xsd:restriction base="xsd:positiveInteger">
                          <xsd:maxExclusive value="100"/> </xsd:restriction>
                      </xsd:simpleType>
                    </xsd:element>
                    <xsd:element name="prix" type="xsd:decimal"/>
                    <xsd:element ref="commentaire" minOccurs="0"/>
                    <xsd:element name="date_livraison" type="xsd:date" minOccurs="0"/>
                </xsd:sequence>
                <xsd:attribute name="num_prod" type="xsd:positiveInteger" use="required"/>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence> </xsd:complexType> </xsd:schema>
```

XPath : l'adressage XML

- XPath
  - Expressions de chemins dans un arbre XML
  - Permet de sélectionner des nœuds par navigation



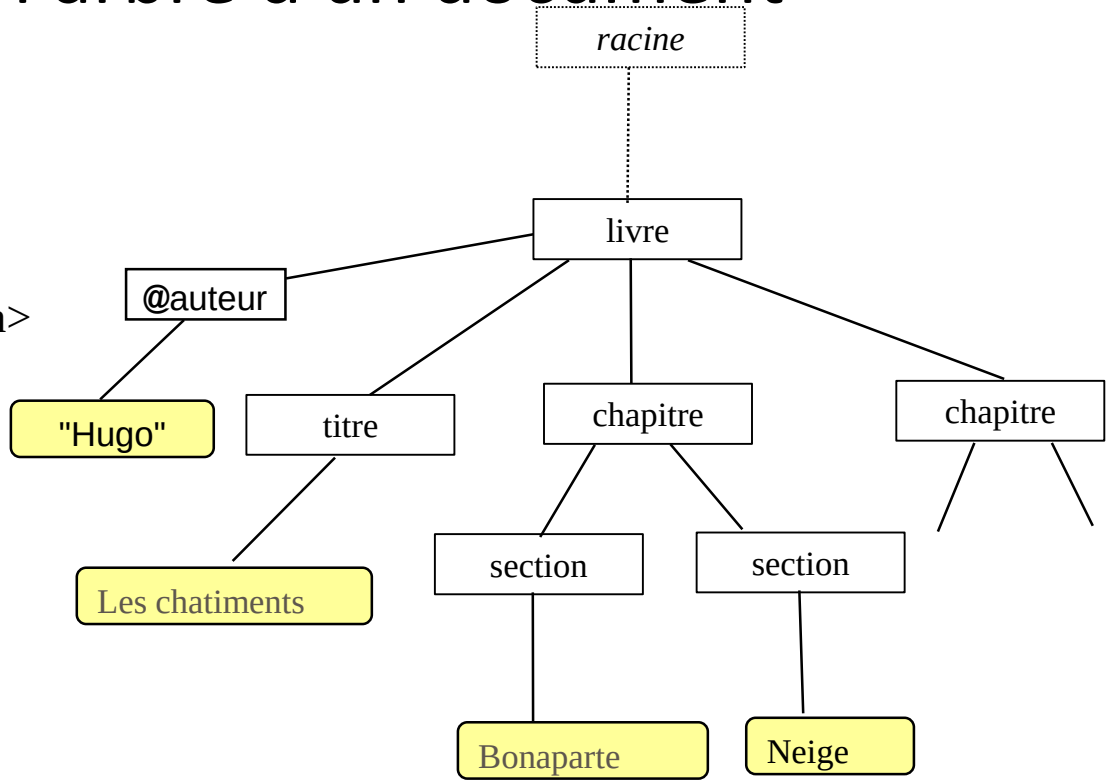
XPath

- XML Path Language
  - recommandation W3C pour expressions de chemins
  - acceptée le 16 novembre 1999
  - version 2 acceptée depuis janvier 2007
- Expressions de chemins communes à :
  - XSL
  - Xpointer (liens)
  - XQuery (queries)
- Xpath permet
  - de rechercher un élément dans un document
  - d'adresser toute sous partie d'un document

XPath - Parcours d'arbre

- XPath opère sur l'arbre d'un document

```
<livre auteur = "Hugo">
  <titre>Les chatiments</titre>
  <chapitre>
    <section>Buonaparte </section>
    <section>Neige</section>
  </chapitre>
  …
</livre>
```



XPath - Expression de chemins

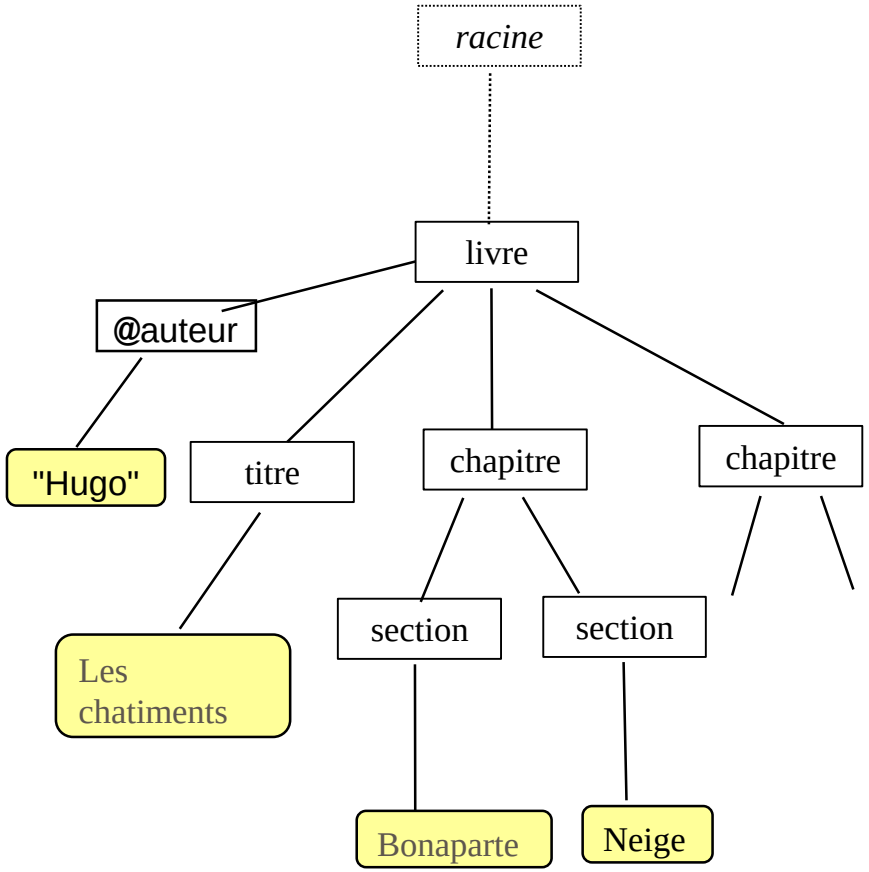
- Une expression de chemins spécifie une traversée de l'arbre du document :
  - depuis un nœud de départ
  - vers un ensemble de nœuds cibles
  - les cibles constituent la valeur du cheminement
- Un chemin peut être :
  - absolu
    - commence à la racine
    - /étape1/…/étapeN
  - relatif
    - commence à un nœud courant
    - étape1/…/étapeN

Syntaxe et sémantique

- Cheminement élémentaire
  - direction::sélecteur [predicat]
- Directions
  - parent, ancestor, ancestor-or-self
  - child, descendant, descendant-or-self
  - preceding, preceding-sibling, following, following-sibling
  - self, attribute, namespace
- Sélecteur
  - nom de nœud sélectionné (élément ou @attribut)
- Prédicat
  - [Fonction(nœud) = valeur]

EXEMPLES

- Sections d'un chapitre
  - /child::livre/child::chapitre/ child::section
  - /livre/chapitre/section
- Texte du chapitre 1 section 2
  - /descendant::chapitre[position() = 1] /child::section[position() = 1] /child::text()
  - //chapitre[0]/section[1]/text()
- Remonter à l'auteur du livre
  - parent::section/ancestor::livre/ child::@auteur
  - ancestor::livre/@auteur



XPath - Synthèse

Pattern	Exemple	Signification
Nom	**section**	Sélectionne les éléments de nom donné
Nom[0]	**section[0]**	Sélectionne le premier élément ayant le nom donné
Nom[end()]	**section[end()]**	Sélectionne le dernier élément ayant un nom donné
|	**Droite|Gauche**	Indique une alternative (un nœud OU bien l'autre (ou les deux))
/	**/**	Sélectionne le nœud racine d'une arborescence
/arbre/Nom	**/livre/chapitre**	Sélectionne les nœuds descendants par la balise de nom donné de l'arbre
*	*	Motif "joker" désignant n'importe quel élément
//	**//personne**	Indique tous les descendants d'un nœud
.	**.**	Caractérise le nœud courant
..	**..**	Désigne le nœud parent. Permet de remonter d'un niveau dans l'arborescence
@	**@nom**	Indique un attribut caractéristique (*@nom* désigne la valeur de l'attribut). La notation *@** désigne tous les attributs d'un élément
text()	**text()**	Désigne le contenu d'un élément (le texte contenu entre ses balises)
ID()	**ID('a2546')**	Sélectionne l'élément dont l'identifiant (la valeur de l'attribut ID) est celui spécifié en paramètre
Comment()	**Comment()**	Désigne tous les nœuds commentaires
Node()	**Node()**	Désigne tous les nœuds

XPath 2.0

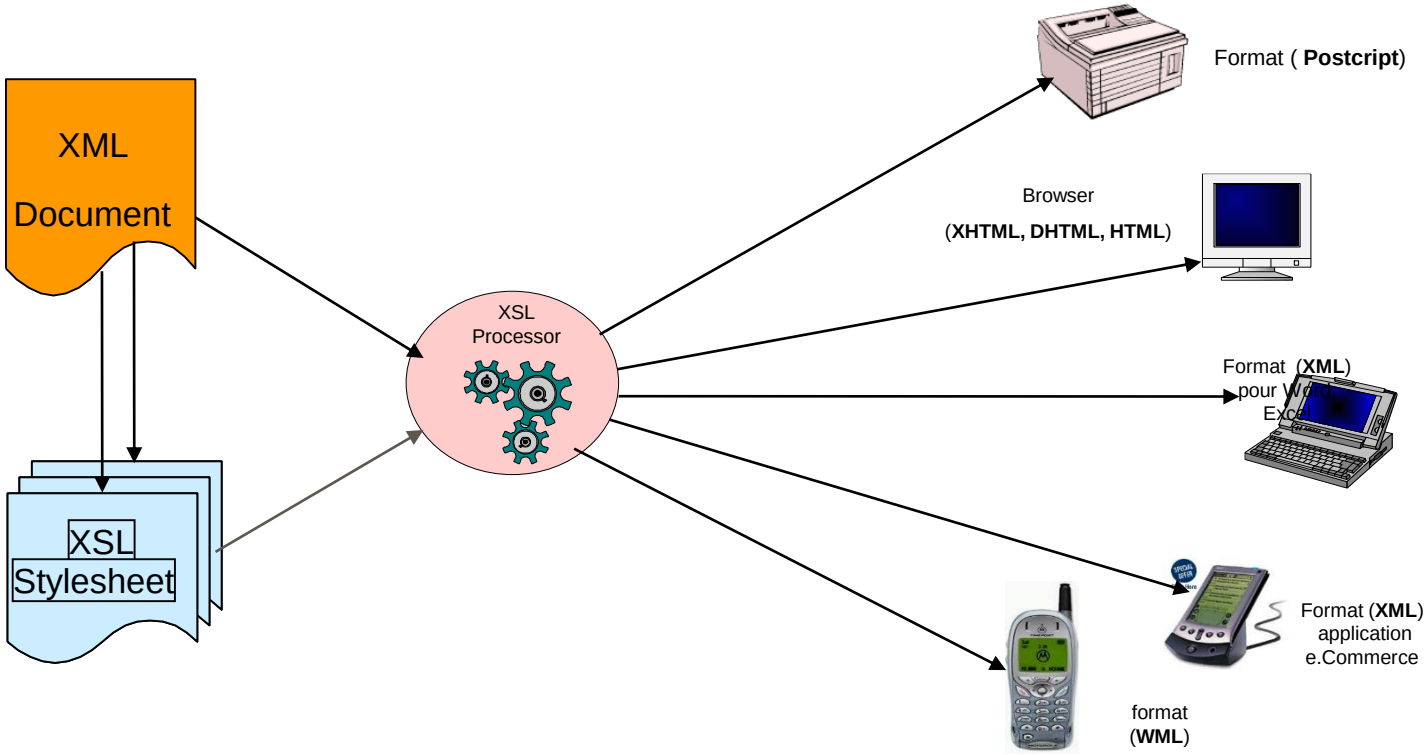
- Types des schémas et séquences supportés
- Elimination des « Result Tree Fragments »
- Support des groupes de noeuds
- Introduction de fonctions d'agrégats
- Support des boucles For et des conditions
- Support des expressions régulières
- Documents multiples en entrée et sortie
- Nouvelles fonctions et opérateurs
- Intégration de fonctions utilisateurs possible

XSLT : la présentation

- Permet de transformer un document
  - régulier ou irrégulier
  - de XML à XML, XHTML est un cas particulier
  - De XML à présentation (HTML, texte, rtf, pdf, etc.)
- Un document est un arbre
- Le processeur XSL parcourt l'arbre et applique les règles de transformations vérifiées (à condition vraie) aux nœuds sélectionnés
- Un document est produit en sortie

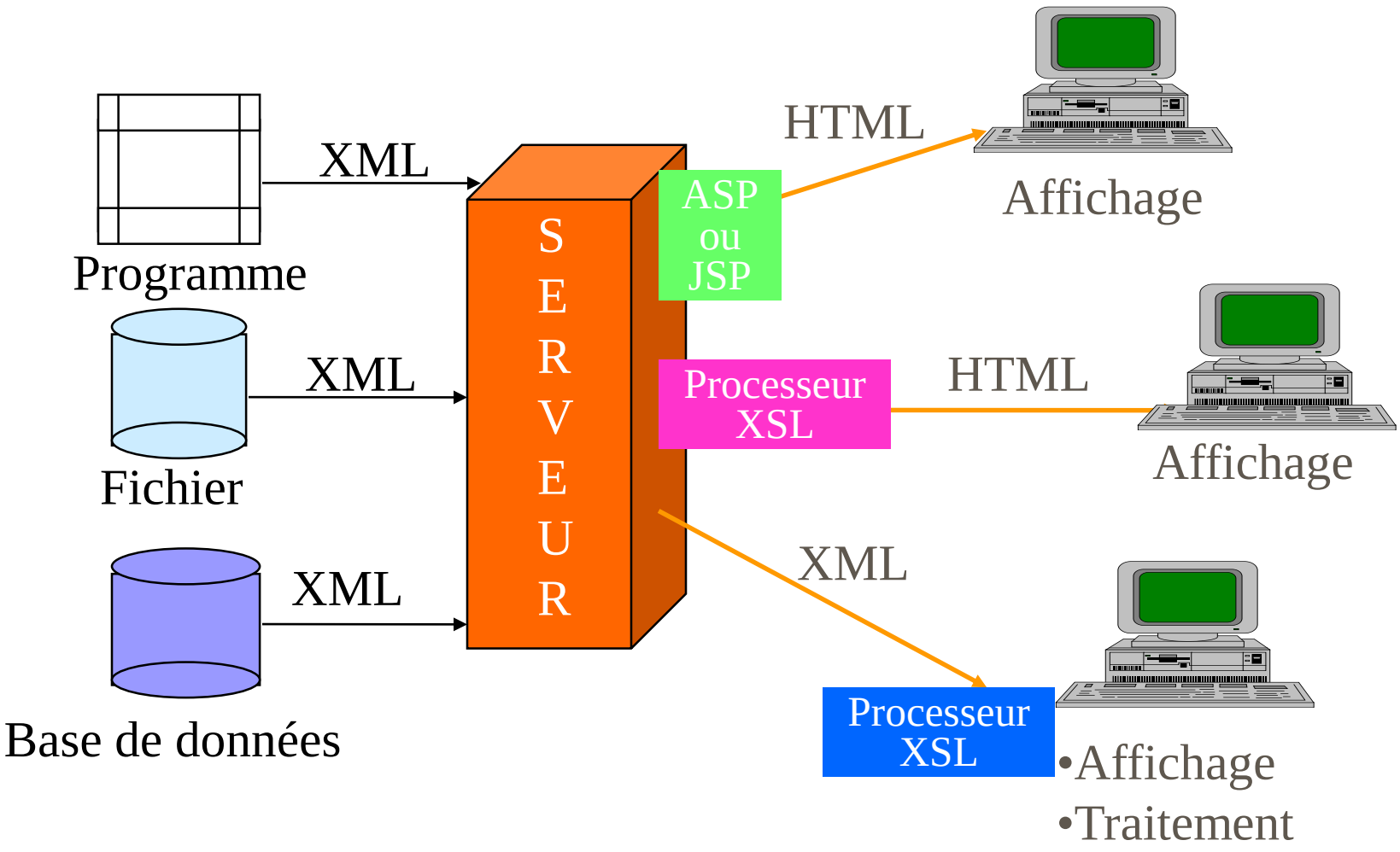
Publications avec XSL

- Plusieurs formats de publication pour un contenu



- XSL permet la présentation sur des terminaux variés

Architectures



Transformation par Jscipt côté client

```
<html> <body>
<script type="text/javascript">
// Load XML
    var xml = new ActiveXObject("Microsoft.XMLDOM")
    xml.async = false
    xml.load("exemple.xml")
// Load XSL
    var xsl = new ActiveXObject("Microsoft.XMLDOM")
    xsl.async = false
    xsl.load("exemple.xsl")
// Transform
    document.write(xml.transformNode(xsl))
</script>
</body> </html>
```

Les feuilles de style

- Une feuille de style XSL
  - est un document XML de racine <xsl:stylesheet>
  - contient une liste de règles de transformation de l'arbre d'entrée en arbre de sortie <xsl:template>
- Chaque règle (<xsl:template>) précise:
  - Une condition spécifiant le sous-arbre du document d'entrée auquel elle s'applique (match=)
  - Une production spécifiant le résultat de l'application de la règle (contenu)
- Il s'agit de règles de production classiques
  - If <condtition> then <production>
  - Codées en XML avec espace de nom xsl:

Exemple de document

```
<?xml version="1.0" ?>
    <Guide>
        <Restaurant Categorie="**">
            <Nom>Le Romantique</Nom>
            <Adresse>
                <Ville>Cabourg</Ville>
                <Dept>Calvados</Dept>
            </Adresse>
        </Restaurant>
        <Restaurant Categorie="***">
            <Nom>Les TroisGros</Nom>
            <Adresse>
                <Ville>Roanne</Ville>
                <Dept>Loire</Dept>
            </Adresse>
        </Restaurant>
</Guide>
```

Exemple de feuille de style XSL

```
<?xml version="1.0" ?>
 <xsl:stylesheet xmlns:xsl="http://www.w3.org/xsl">
 <xsl:template match="/">
       <html><head><B>ESSAI XSL</B></head>    <body><xsl:apply-
     templates/></body></html>
  </xsl:template>
 <xsl:template match="Guide">
        <H1>BONJOUR LE GROUPE XML</H1>
        <H2>SUIVEZ LE GUIDE</H2>
<xsl:apply-templates />
</xsl:template>
<xsl:template match="Restaurant">
     <P> <I>Restaurant :</I>
     <xsl:value-of select="Nom"/></P>
</xsl:template>
</xsl:stylesheet>
```

Les règles de production

- Définition des règles par <xsl:template …>
- Attributs
  - *match*: condition de sélection des nœuds sur lesquels la règle s'applique (XPath)
  - *name*: nom de la règle, pour invocation explicite (en conjonction avec <call-template>)
  - *priority*: priorité, utilisé en cas de conflit entre deux règles ayant la même condition
- Exemples
  - <xsl: template match="/">
  - <xsl: template match="auteur" name="R1" priority="1">

La génération du résultat

- Le contenu de l'élément <xsl:template> est la production:
  - Les éléments du namespace xsl sont des instructions qui copient des données du document source dans le résultat
  - Les autres éléments sont inclus tels quels dans le résultat
- Instructions pour:
  - Parcourir l'arbre du document source
  - Copier le contenu du document source dans le résultat
- Parcours de l'arbre:
  - <xsl:apply-templates>, <xsl:for-each>
- Copie du contenu du nœud sélectionné:
  - <xsl:value-of select= … >

Résumé des commandes

- <xsl:template>
  - définir une règle et son contexte
- <xsl:apply-templates />
  - appliquer les transformations aux enfants du nœud courant
- <xsl:value-of select … />
  - extrait la valeur d'un élément sélectionné à partir du nœud courant
- <xsl:for-each>
  - définir un traitement itératif
- <xsl:pi>
  - générer une instruction de traitement
- <xsl:element>
  - générer un élément
- <xsl:attribute>
  - générer un attribut
- <xsl:if>
  - définir un traitement conditionnel