XML

Extensible Markup Language

Plan

- XML et XSD : langage universel et standard
- Les namespaces et les types
- Le langage et les requêtes Xpath
- Les transformations XLST

L'origine du XML

- e**X**tensible **M**arkup **L**anguage
- Langage à balise basé sur SGML (Standard Generalized Markup Language)
- Normé depuis 1998 par le W3C
- Langage permettant de séparer la sémantique de l'information même
- Extensible grâce au système de namespaces et de création de grammaire

Format XML

- Déclaration XML

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

- Blocs délimités par des balises
  - Ouvrante et fermante
  - Balise simple

```
<complex-tag>
    <simple-tag />
</complex-tag>
```

- Attributs
  - Disponibles dans les balises ouvrantes et simples
  - Associent une clé et une valeur pour l'élément courant

```
<tag attribute="value">
    ...
</tag>
```

- Commentaires

```
<tag />
<!-- Comment in the XML code -->
```

- Caractères spéciaux

```
<tag>Text with special characters "&eacute; &agrave; &amp;"</tag>
```

Langage normé

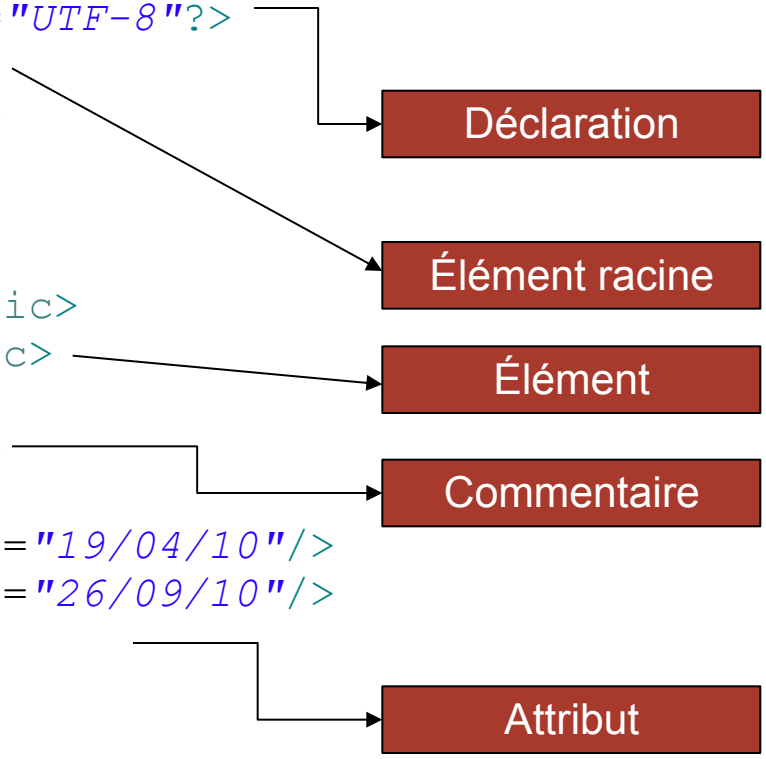
- Langage orienté sur la séparation de contenu et de valeur sémantique
- Nécessité d'imposer des normes strictes
- Normes définies par le W3C
- Tout document XML respecte une syntaxe commune
- Possibilité de valider le XML en général
  - http://validator.w3.org/
  - http://www.w3schools.com/xml/xml_validator.asp
- A terme l'intérêt est de spécifier encore plus la syntaxe pour obtenir un document clair et concis

Utilité du langage

- Langage servant de pont entre humains et machines
  - Structuré
  - Lisible
  - Interopérable (basé sur du texte ascii et non exécuté)
- Selon les besoins peut être parsé ou écrit facilement manuellement ou automatiquement
- Possibilité de manipulation, requêtes et transformation de contenu
  - Transformation en XML selon le contenu
  - Transformation en binaire
  - Requêtes par des langages tel que XQuery/XPath ou XQL

Document XML

```
<?xml version="1.0" encoding="UTF-8"?>
<formation>
  <titre>Web Services</titre>
  <durée>3</durée>
  <!-- liste des publics -->
  <publics>
    <public>Développeur</public>
    <public>Architecte</public>
  </publics>
  <!-- liste des sessions -->
  <sessions>
    <session du="17/04/10" au="19/04/10"/>
    <session du="24/09/10" au="26/09/10"/>
  </sessions>
</formation>
```



Un document XML bien formé (1/2)

- L'élément XML peut être représenté par
  - Une balise ouvrante (ie. <balise>) et une balise fermante (ie. </balise>)

```
<titre>Web Services</titre>
```

  - Une balise vide (ie. <balise/>)

```
<session du="17/04/10" au="19/04/10"/>
```

- La structure balise ouvrante/balise fermante peut encadrer
  - Du texte
  - D'autres éléments
  - Un mélange de texte et d'éléments

Un document XML bien formé (2/2)

- Un balise ouvrante ou vide peut contenir des attributs

```
<session du="17/04/10" au="19/04/10"/>
```

- Les balises doivent être correctement imbriquées

```
<Italique><Gras>Document mal formé</Italique></Gras>

<Italique><Gras>Document bien formé</Gras></Italique>
```

- Remarque : XML est sensible à la casse

XSD

- XSD = XML Schema Definition
- Un schéma XML a pour objectif de définir la structure d'un document XML
- XSD est le successeur de DTD
- Un schéma XML définit
  - Les éléments et attributs pouvant apparaître
  - La hiérarchie et l'ordre des éléments du document
  - Si un élément peut/doit contenir du texte
  - Les types des éléments (ex : chaîne, entier …)
  - …

Exemple XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.zenika.com/disques"
   xmlns:tns="http://www.zenika.com/disques"
   elementFormDefault="qualified">

      <element name="disque" type="tns:Disque"/>

      <complexType name="Disque">
            <sequence>
                  <element name="titre" type="string"/>
                  <element name="auteur" type="string"/>
            </sequence>
      </complexType>

</schema>
```

Les namespaces (1/3)

- Namespace = « espace de nommage »
- Un namespace a pour objectif d'assurer l'unicité d'un élément XML

```xml
<disque>
  <titre>Beautiful Freak</titre>
  <auteur>Eels</auteur>
</disque>
```

```xml
<disque ref="CD_0001">
  <titre>Road To Ruin</titre>
  <artiste>Ramones</artiste>
  <annee>1979</annee>
</disque>
```

- La convention est d'utiliser une URI pour définir un namespace
  Exemple : http://www.zenika.com

Les namespaces (2/3)

- Déclaration d'un namespace ► **xmlns:***prefix*="*namespace*"

```
<ns:disque xmlns:ns="http://www.zenika.com/disques">
```

- Association d'un élément à un namespace

```
<ns:disque xmlns:ns="http://www.zenika.com/disques">
  <ns:titre>Beautiful Freak</ns:titre>
  <ns:auteur>Eels</ns:auteur>
</ns:disque>
```

- Possibilité de déclarer plusieurs namespaces

```
<cd:disque ref="CD_0001"
    xmlns:cd="http://www.cdiscues.fr"
    xmlns:txt="http://www.cd-text.fr">
  <cd:titre>Road To Ruin</cd:titre>
  <txt:artiste>Ramones</txt:artiste>
  <txt:annee>1979</txt:annee>
</cd:disque>
```

Les namespaces (3/3)

- Définition d'un namespace par défaut ► **xmlns="***namespace***"**

```
<disque xmlns="http://www.zenika.com/disques">
  <titre>Beautiful Freak</titre>
  <auteur>Eels</auteur>
</disque>
```

- La définition d'un/de namespace(s) est optionnelle mais est fortement conseillée

- Si aucun namespace n'est défini, les éléments sont associés au namespace par défaut

- Il est possible d'associer un namespace aux attributs
  - Même namespace que l'élément père
  - Autre namespace que l'élément père

Des éléments et des types

- Il existe des types pré existants, considérés comme simples
  - **xs:string**
  - **xs:decimal**
  - **xs:integer**
  - **xs:boolean**
  - **xs:date**
  - **xs:time**
- Une fois un type choisi il est possible de le restreindre grâce à d'autres balises spécifiées dans les restrictions

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:simpleType name="simpleType">
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="120"/>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

Les attributs XSD

- Une fois les éléments et leurs types déclarés, il est possible de spécifier des attributs pour chaque élément
- Les attributs sont accompagnés d'un type simple et sont déclarés directement dans l'élément
- Comme avec DTD, il est possible de leur assigner une valeur par défaut ou les rendre obligatoires

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="elementType">
        <xs:attribute name="attribute-name" type="xs:string" />
        <xs:attribute name="attribute-name2" type="xs:string" default="test" />
        <xs:attribute name="attribute-name3" type="xs:string" use="required" />
    </xs:complexType>
</xs:schema>
```

Faire le bon choix

- Quel système choisir entre DTD et XSD ?
  - DTD
    - *Hérité de SGML, compatible avec d'anciens systèmes, largement porté*
    - *Syntaxe légère mais obsolète et incomplète*
  - XSD
    - *Système spécifique pour le XML, possède beaucoup plus de capacités pour restreindre et spécifier une grammaire*
    - *Les espaces de nom permettent d'utiliser différents XSD en même temps*
- Il existe encore d'autres alternatives dont une des plus fameuse est RELAX-NG, qui est plus concise mais plus jeune
- Tout dépend des besoins de l'application

XPATH – Langage de requêtes XML

- En l'état, le XML ne permet pas de faire de requêtes pour récupérer des éléments précis
- C'est pour cela que XPath a été développé
- Standard créé par le W3C en 1999

- Le but est de pouvoir sélectionner des parties de document
- Il est possible de naviguer dans un fichier XML
- L'idée est de filtrer les données par de simples opérations sur les éléments XML

Fonctionnement par noeuds

- Comme pour les parseurs XML, XPath se base sur des nœuds
  - Root, la racine du document
  - Element, une balise
  - Text, du contenu textuel
  - Attribute, un attribut d'une balise
  - Comment, commentaire XML

XML d'exemple

```xml
<?xml version="1.0" encoding="utf-8"?>
<customers>
    <customer>
        <fullname usual-name="Steve">Steven O'daily</fullname>
        <nationality>irish</nationality>
        <age>45</age>
        <child>
            <fullname usual-name="Tommy">Thomas O'daily</fullname>
            <age>13</age>
        </child>
    </customer>
    <customer>
        <fullname>Francis Pirelli</fullname>
        <nationality>italian</nationality>
        <age>17</age>
    </customer>
    <customer>
        <fullname usual-name="Angie">Angelina Ferrando</fullname>
        <nationality>spanish</nationality>
        <age>19</age>
    </customer>
    <customer>
        <fullname>Karina Drovodek</fullname>
        <nationality>russian</nationality>
        <age>34</age>
        <child>
            <fullname usual-name="Vlad">Vladimir Drovodek</fullname>
            <age>3</age>
        </child>
    </customer>
    <customer>
        <fullname usual-name="Tony">Antonio Tival</fullname>
        <nationality>italian</nationality>
        <age>24</age>
    </customer>
</customers>
```

Recherche par location 1/2

- La requête XPath est séparée en plusieurs étapes par des "/"
- Chaque étape a une valeur définissant le/les nœuds sélectionnés
  - / : Sélection depuis la racine
    - *"/customers" sélectionne la balise customers à la racine*
  - // : Sélection dans tous les descendants
    - *"//age" sélectionne tous les tags "age"*
  - *NomDeNoeud* : sélection de l'élément
    - *"/customers/customer" sélectionne tous les tags customer dans la balise racine* customers

Recherche par location 2/2

- . : Sélection du nœud courant
  - *"/customers/customer/." fait la même chose qu'au dessus*
- .. : Noeud parent
  - *"//age/.." sélectionne tous les nœuds ayant un sous nœud "age"*
- @ : Attribut
  - *"//@usual-name" sélectionne tous les attributs usual-name*

Quelques exemples

- //child/..
  - Sélectionne les clients ayant un enfant
- //@usual-name/../../nationality
  - Sélectionne la nationalité des clients ayant un nom d'usage
- /customers/customer/age
  - Sélectionne l'age de tous les clients (sans les enfants)
- //child/age
  - Récupère l'age de tous les enfants
- //child/../fullname/@usual-name
  - récupère les noms d'usage des personnes ayant un enfant

Une sélection sous condition

- Il est possible de conditionner les sélections grâce au crochets

- Les prédicats sont
  - [1] : Premier élément correspondant
    - *//child[1] premier enfant du document*
  - [@attribute] : Ayant l'attribut donné
    - *//customer[@usual-name] clients ayant un nom d'usage*
  - [age>12] : Ayant un âge supérieur à 12
    - *//customer[age>21] personnes âgées de plus de 21 ans*

- Les conditions peuvent être utilisés avec les opérateurs suivant
  - | + - * div = != < <= > >= or and mod

Quelques exemples

- //child[../fullname/@usual-name]
  - Sélectionne les enfants ayant pour parent une personne avec un nom d'usage
- //@usual-name[../../nationality='italian']
  - Sélectionne le nom d'usage des personnes italiennes
- //child/age[../../nationality!='irish']
  - Sélectionne l'age des enfants ayant un parent non irlandais
- //customer[age>30][age<4*child/age]
  - Sélectionne les clients de plus de 30 ans ayant au plus 4 fois l'age de leur enfant

Des axes

- Il est aussi possible de se déplacer sur différents axes
  - ancestor:: – Les ancêtres d'un nœud
  - ancestor-or-self:: – Les ancêtres et soit
  - attribute:: – Les attributs du nœud
  - child:: – Les enfants du nœud
  - descendant:: – Les enfants/petits enfants du nœud
  - descendant-or-self:: – Les descendants du nœud et soit
  - following:: – Tous les nœuds "après" le nœud courant
  - following-sibling:: – Les nœuds frères après le nœud courant
  - parent:: – le parent du nœud courant
  - preceding:: – Les nœuds précédents
  - preceding-sibling:: – Les nœuds frères précédents
  - self:: – Le nœud courant

Quelques exemples

- //customer[nationality='italian']/following-sibling::customer[1]
  - Sélectionne le premier client après chaque italien

- //customer[1]/descendant::fullname
  - Séctionne tous les fullname situés dans le dossier du premier client (enfants inclus)

Des combinaisons et méthodes

- Lorsque l'on veut sélectionner quelque chose pouvant répondre à deux conditions différentes, l'opérateur « | » permet de faire une union des deux résultats

- Méthodes
  - Sur les nœuds

    count(), last(), name(), position(), text()
  - Sur les chaines

    concat(a, b), contains(a, b), start-with(a, b), string-length(a), …
  - Sur les nombres

    ceil(a), floor(a), sum(a, b, ...)

XSLT, pourquoi ?

- Il est parfois intéressant de faire une transformation du contenu XML en un autre

- Changer un document XML en XHTML
- Convertir un format XML vers un autre
- Filtrer des données ne devant pas figurer sur un autre format XML

- XSLT, eXtensible Stylesheet Language Transformations est fait pour ces opérations délicates

Fichier XLST (XLS template)

- Le fichier XSLT doit commencer par la balise racine <xsl:stylesheet> ou <xsl:transform>, les deux étant synonymes
- Le contenu sera une suite de templates appliqués sur des sections du fichier XML pris en entrée

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/customers/customer[1]">
  <html>
  <head><title><xsl:value-of select="fullname/@usual-name" /></title></head>
  <body>
      <h1>Customer</h1>
      Name: <xsl:value-of select="fullname/text()" /><br />
      Age: <xsl:value-of select="age/text()" /><br />
      Nationality: <xsl:value-of select="nationality/text()" />
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Définition d'un template principal

- Le template principal sélectionne une portion du document XML à convertir et fourni un résultat en réintégrant les valeurs du fichier XML

- L'insertion de données se fait avec l'aide de quelques balises
  - <xsl:value-of select="xpath">
    - *Affiche une valeur*
  - <xsl:for-each select="xpath">
    - *Boucle sur les résultats de sélection*
  - <xsl:if test="expression">
    - *Execute une partie de template si le teste est réussi*
  - <xsl:choose> : Permet de faire une condition et son opposée
    - *<xsl:when test="expression"> : un cas du choose*
    - *<xsl:otherwise> : cas par défaut*

Exemple complet de transformation

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/customers">
    <html>
    <head><title>Customers list</title></head>
    <body>
        <h1>Customers</h1>
        <xsl:for-each select="customer">
            <div>
                Name: <xsl:value-of select="fullname/text()" /><br />
                Age: <xsl:value-of select="age/text()" /><xsl:if test="age/text() &lt;
21"> Not major in every country</xsl:if><br />
                Nationality: <xsl:value-of select="nationality/text()" /><br />
                <xsl:choose>
                    <xsl:when test="child">
                        <strong>Has a child</strong><br />
                        Name: <xsl:value-of select="child/fullname/text()" /><br />
                        Age: <xsl:value-of select="child/age/text()" />
                    </xsl:when>
                    <xsl:otherwise>
                        <strong>Hasn't a child</strong>
                    </xsl:otherwise>
                </xsl:choose>
            </div>
            <hr />
        </xsl:for-each>
    </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

Délégation de tâche

- Lorsqu'un seul template devient important, il est rapidement impossible de le maintenir
- Une option permet de découper le template principal en appliquant des sous templates

- **<xsl:apply-templates/>** est utilisé pour invoquer d'autre templates
- Lors d'un apply-templates, tout template matchant l'élément courant sera invoqué

- Il est possible de changer l'élément actuel uniquement pour le apply-template via l'attribut **select**

Templates par l'exemple

- Voici l'utilisation d'un second template pour déléguer l'affichage d'un enfant

```
<xsl:template match="child">
    <strong>Has a child</strong><br />
    Name: <xsl:value-of select="fullname/text()" /><br />
    Age: <xsl:value-of select="age/text()" />
</xsl:template>
```

- Pour faire appel à ce template, il suffit remplacer l'ancien bloc par :

```
<xsl:apply-templates select="child" />
```