eXtensible Markup Language

XML en bref — Partie I

Date de la dernière modification : 15 septembre 2009

Plan

Introduction à XML

- Pourquoi XML ?
- Mon premier document XML
- Filiation et héritage
- Exemples de dialectes
- Les avantages

[diapos de Serge Abiteboul, Univ. Paris Sud]

Analyse

La situation :

- Révolution Internet et HTML
- Données massivement distribuées
- Développement des applications de *e-business*

Les faits :

1. Défaut de typage et structuration pour HTML
2. Bases de données ?
   - *La structure* implicite, irrégulière, partielle, indicative
   - *Le schéma* a posteriori, complexe, ignoré, fluctuant, évolutif

La solution : **Modèle de données semi-structurées**
Gestion de documents & Gestion de données $\longrightarrow$ XML

Exemple de document XML

Modèle d'*arbre* et langage à *balises* : syntaxe, structure

```
<livre>
    <titre>Une saison de machettes</titre>
    <auteur>
        <prenom>Jean</prenom>
        <nom>Hatzfeld</nom>
    </auteur>
    <edition>Seuil</edition>
    <date>22 août 2003</date>
</livre>
```

Sémantique absente (et comportement et traitement)

`<a>tarte</a>` : dessert ? coup ?

`<dessert>tarte</dessert>` pas mieux

Filiation & héritage

≪ HTML killer ≫ (Tim Berners Lee)

- ensemble fini et normalisé (4.0) de balises de présentation `center`, `hr`, `b`, `i`, *etc.*
- rendu dépendant du butineur
- multiples versions pour présentations variées
- indexation textuelle

≪ SGML light ≫ (*Standardized Generalized Markup Language*)

- Métalangage de description et d'échange de documents structurés
- Balisage structurel
- documentation technique en milieu industriel

XML = 10% SGML pour couvrir la plupart des applications

Les applications

Démarche générale pour la création d'un dialecte XML :

1. Définir la syntaxe : vocabulaire et grammaire du langage
2. Préciser la sémantique : ça veut dire quoi `<a>tarte</a>` ?

- XHTML : du HTML xml-isé
- MathML : expressions mathématiques
- BibTeXML : références bibliographiques
- SVG : graphiques vectoriels 2D
- XSL : feuilles de style pour XML
- SOAP : calcul distribué par appels de services distants
- WSDL : description de services web
- XML Schema : typage de documents XML
- XMLResume, CuisineML, TrucInsignifiantML, *etc.*

XML c'est bien parce que...

- développement et promotion par le World Wide Web Consortium (W3C)
- simplicité de production, de lecture, d'analyse et d'interprétation
- un seul document (données) pour de multiples supports et applications
- partage d'information dans des communautés d'intérêt
- modularité et réutilisation des langages
- interopérabilité des outils de traitement
- interrogation et échange de données hétérogènes

Le langage XML

La syntaxe XML définit la représentation sérialisée d'un arbre :

- les nœuds (document, élements, commentaires, attributs, . . .)
- la structure (e.g. imbrication des balises)

Un document XML comprend trois parties :

1. un *prologue* (optionnel), avec la déclaration XML, la DTD, des commentaires, des instructions de traitement
2. un *élément racine* avec son contenu
3. un *épilogue* (optionnel), avec des commentaires, des instructions de traitement

Le contenu du document proprement dit est le contenu de l'élément racine

[diapos de Bernd Amann, ENST]

Un exemple

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- description d'un livre -->
<livre isbn="2020525712" nbre_pages="233">
    <titre>Allah n'est pas obligé</titre>
    <auteur>
        <prenom>Ahmadou</prenom>
        <nom>Kourouma</nom>
    </auteur>
    <edition>Seuil</edition>
    <date>
        <mois>septembre</mois>
        <annee>2000</annee>
    </date>
    <resume>
        <![CDATA[ contenu <b>non analysé</b> ]]>
    </resume>
</livre>
```

La syntaxe

- La déclaration
- Les éléments
- Les attributs
- Les commentaires et les instructions de traitement
- Les sections littérales
- Les entités

Les DTDs

Document Type Definition

- Document XML **bien formé** : imbrication des éléments
- Document XML **valide** : conformité à une DTD et intégrité référentielle
- Une DTD spécifie la *grammaire* d'un document XML valide
- Une DTD *n'est pas* un document XML
- Modèle de contenu des éléments :
  - simple : `ANY EMPTY (#PCDATA)`
  - complexe : basé sur des expressions régulières `,.+*?|()`
- 2 usages :
  - déclaration interne au document XML valide
    `<!DOCTYPE albums [ ... ]>`
  - définition externe et référence dans tout document XML valide
    `<!DOCTYPE albums SYSTEM "albums.dtd">`

[diapos de Bernd Amann, ENST]

albums.dtd

```
<!ELEMENT albums (groupe*, album*) >
<!ELEMENT groupe (nom, membre+) >
<!ATTLIST groupe gid ID #REQUIRED >
<!ELEMENT nom (#PCDATA) >
<!ELEMENT membre (#PCDATA) >
<!ELEMENT album (titre, interprète, année?) >
<!ELEMENT titre (#PCDATA) >
<!ELEMENT interprète EMPTY >
<!ATTLIST interprète iid IDREF #REQUIRED >
<!ELEMENT année (#PCDATA) >
```

Limitations des DTDs

- ordre des nœuds imposé (schéma relationnel)
- langage dédié ($\neq$ XML)
- expression de contraintes sur le contenu
- typage indépendant du contexte : 1 balise = 1 type
  solution partielle : un espace de noms par élément
  (*auteur.nom* et *étudiant.nom*)
- contraintes et portée des `IDREFs`

Conclusion sur les DTDs :
Compacte et simple à valider, mais pas encore digne d'un schéma de BD
Il existe des langages *plus riches* pour la description d'un document XML (à suivre...)

XML et la programmation

Application Programming Interfaces (API) pour chaque couche :

1. analyse syntaxique (*parsing*) **SAX**/XNI
2. validation
3. chargement **DOM**/JDOM
   - manipulation d'un document en mémoire
   - sauvegarde d'un document
4. traitement complexe
   - sérialisation (*binding*) JAXB/CASTOR/ZEUS
   - transformation d'un document XML TrAX
   - interprétation d'un dialecte XML : SVG, MathML, etc.
5. *etc.*

API transverse : *Java API for XML Processing* (JAXP)

[diapos de Fabrice Rossi, Univ. Paris IX Dauphine]

Les API SAX et DOM

L'API DOM (*Document Object Model*) :

- Construit une représentation du document en mémoire sous forme d'arbre
- Adaptée aux applications qui modifient ou traitent dans leur globalité un document
- Peu performant (emprunte mémoire et coût temporel)

L'API SAX (*Simple API for Xml*) :

- Définit des triggers qui se déclenchent sur certaines balises
- Adaptée aux applications qui extraient de l'information d'un document
- Délicat à programmer et très bas niveau

Limitations de l'API SAX

- accès séquentiel : perte de la structure (contexte)
- pas de mémoire globale : accumulation et restitution impossibles

Solutions possibles :

- déterminer le modèle de l'analyse (par ex. un automate)
- conserver les résultats intermédiaires dans de nouvelles variables
- écrire de nouvelles procédures pour obtenir les résultats

Au-delà des DTDs

- Le typage n'est pas imposé dans XML mais il facilite la navigation, l'interrogation, l'optimisation et il permet l'interopérabilité
- Projet ISO DSDL (*Document Schema Definition Language*)
- Classification des langages d'écriture de schéma XML :
  - ouvert vs. fermé
  - à base de règles / à base de grammaire / orienté objet
  - types-balises couplés vs. découplés
- Les favoris : DTD, XML Schema, Relax NG, Schematron

XML Schema

- 44+ types de données natifs (vs. 10 pour les DTDs)
- Extension des types de données (paramétrage par aspects)
  Exemple : `'\p{Sc}\p{Nd}+(\.\p{Nd}\p{Nd})'` (`$99.99`)
- Syntaxe XML : pas de nouveau langage à apprendre
- Typage « orienté objet » :
  par extension ou restriction de type (dérivation)
- Expression d'ensembles (ordre quelconque, balise `all`)
- Définition fine de la portée des identifiants (clefs)
- Découplage entre type et balise :
  différents modèles de contenu pour une même étiquette
- Substitution d'éléments :
  attribut `substitutionGroup` avec type identique ou dérivé
- Gestion des espaces de noms

[diapos de David Laurent, Univ. Evry]

XML Schema

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.bibliotheque.org"
        xmlns:bb="http://www.bibliotheque.org"
    <element name="Bibliothèque">
        <complexType>
            <sequence>
                <element ref="bb:Livre" maxOccurs="unbounded"/>
            </sequence>
        </complexType>
   </element>
    <element name="Livre">
        <complexType>
            <sequence>
                <element ref="bb:Titre"/>
                <element ref="bb:Auteur"/>
                ...
            </sequence>
        </complexType>
    </element>
    <element name="Titre" type="string"/>
    <element name="Auteur" type="string"/>
    ...
</schema>
```

Relax NG

La structure d'un schéma reproduit l'exemple d'un document valide

- Définition de motifs
  - `element` et `attribute`
  - `define` et `ref` avec `grammar` et `start`
- Quelques propriétés du langage :
  - \+ simple et modulaire (`externalRef-ref` et `include-ref`)
  - \+ typage indépendant des balises
  - \+ gestion des espaces de noms
  - \+ dialecte XML
  - \+ modèle de contenu évolué (`group`, `choice`, `interleave`)
  - +/− extensibilité limitée
    - types de données de base : `text`, `choice` avec `value`, ou liste
    - bonus : types définis en dehors de la spec. Relax NG (`data-type` et `datatypeLibrary`)
  - − normalisation "faible" : Consortium OASIS

[diapos de Fabrice Rossi, Univ. Paris IX Dauphine]

catalogue-articles.rng

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<element name="catalogue"
        xmlns="http://relaxng.org/ns/structure/1.0"
        ns="http://gr.org/alxml/catalogue-articles"
        datatypeLibrary="path/to/XMLSchema-datatypes">
    <oneOrMore>
        <element name="article">
            <element name="description"><text/></element>
            <element name="prix">
                <data type="float">
                    <param name="minExclusive">0</param>
                </data>
            </element>
            <externalRef href="nomenclature.rng"/>
        </element>
    </oneOrMore>
</element>
```

Curieux ? Un exemple avec Schematron

```
<schema xmlns="http://www.ascc.net/xml/schematron" >
     <pattern name="Modèle ouvert (avec résultats négatifs et positifs)">
          <rule context="AAA">
               <assert test="BBB">élément BBB manquant</assert>
               <report test="BBB">élément BBB présent</report>
               <assert test="CCC">élément CCC manquant</assert>
               <report test="CCC">élément CCC présent</report>
          </rule>
     </pattern>
     <pattern name="Modèle fermé (avec résultats négatifs seulement)">
          <rule context="AAA">
               <assert test="BBB">élément BBB manquant</assert>
               <assert test="CCC">élément CCC manquant</assert>
               <assert test="count(BBB|CCC) = count (*)">il y a trop d'éléments</assert>
          </rule>
     </pattern>
     <pattern name="Caractère @ interdit">
          <rule context="*">
               <report test="contains(.,'@')">
                   Le texte de l'élément <name/> ne doit pas contenir le caractère @
               </report>
          </rule>
     </pattern>
</schema>
```

XPath

XPath est un langage d'extraction de nœuds dans un arbre XML :

- un chemin de navigation est une séquence d'étapes
- à chaque étape : un **axe**, un **filtre** et des **prédicats** optionnels
- le résultat d'une étape est une séquence de nœuds

$[/]étape_1/axe::filtre[prédicat_1]...[predicat_n]/étape_2/.../étape_k$

Une expression XPath :

- s'évalue en fonction d'un **nœud contexte**
- désigne un ou plusieurs chemin-s dans l'arbre à partir du nœud contexte

[diapos de Bernd Amann, ENST]

Quelques expressions de chemins

- `/A/B[@att1]`
  Les nœuds `/A/B` qui ont un attribut `@att1`
- `/A/B[@att1='a1']`
  Les nœuds `/A/B` qui ont un attribut `@att1` valant `'a1'`
- `/A/B/descendant::text()[position()=1]`
  Le premier nœud de type `Text` descendant d'un `/A/B`
- `/A/B/descendant::text()[1]`
  idem
- `FILM[@TITRE='Brazil' and ACTEUR/NOM='De Niro']`
  Le film Brazil avec l'acteur Robert De Niro
- `FILM[@TITRE='Brazil'][ACTEUR/NOM='De Niro']`
  idem
- `FILM[not(ACTEUR[NOM='Willis'])]`
  Les films sans l'acteur Bruce Willis

XSL-T

Extensible Stylesheet Language Transformations

- Transformation XSL :
  production d'un **document cible** quelconque (XML, PDF, TXT, LaTeX, etc.)
  à partir d'un **document XML source**
- Transformation d'arbre XML par :
  - extraction de données
  - génération de contenu
  - suppression de contenu (nœuds)
  - Ddéplacement de contenu (nœuds)
  - duplication de contenu (nœuds)
  - tri
- Programme XSLT : ensemble de **règles** pour construire un résultat

[diapos de Bernd Amann, ENST]

Les règles XSL-T

- Élément de base pour produire le résultat : `template`
- Une **règle** s'applique dans le contexte d'un **nœud** de l'arbre, identifié par le **motif** (`match`) de la règle
- Déclenchement d'une règle
  (avec `xsl:apply-templates` et sur un nœud courant) :
  1. vérification du `mode` (identique)
  2. test d'appariement du motif (`match`) avec le nœud courant
  3. choix de la règle de priorité maximale
- Exécution du programme XSLT (appel des règles) :
  1. règle initiale appliquée sur '/'
  2. production du cadre du document cible
  3. appel d'autres règles pour compléter le résultat
- L'application d'une règle produit un **fragment du résultat**

films-to-html.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="FILMS">
    <html>
        <head><title>Liste des films</title></head>
        <body bgcolor="white">
            <xsl:apply-templates select="FILM" mode="ancres"/>
            <xsl:apply-templates select="FILM"/>
        </body>
    </html>
</xsl:template>

<xsl:template match="FILM" mode="ancres">
    <a href="#{TITRE}">
        <xsl:value-of select="TITRE"/>
    </a>
</xsl:template>

 <xsl:template match="FILM">
    <a name="{TITRE}"/>
    <h1><xsl:value-of select="TITRE"/></h1>
    <b><xsl:value-of select="ANNEE"/>,</b>
    <xsl:value-of select="GENRE"/>
    <br/>
    <b>Réalisateur</b> : <xsl:value-of select="REALISATEUR"/>
</xsl:template>
</xsl:stylesheet>
```