eXtensible Markup Language

XML en bref — Partie II

Guillaume . Raschia @univ-nantes.fr

Polytech Nantes, Dpt. INFO

Date de la dernière modification: 10 octobre 2013

Rappel des épisodes précédents

- Le monde en XML
- Éléments de syntaxe du (méta-)langage XML
- Créer des dialectes : les DTD
- Modèles de programmation pour XML : DOM & SAX
- Encore du typage : Relax NG, XML Schema & Schematron
- Trouver son chemin : XPath
- Transformer avec du style : XSL-T

Une histoire d'arbres

Un document XML est vu comme une structure

- arborescente (limitation : IDREF's)
- finie
- ordonnée (limitation : attributs)
- étiquetée sur un alphabet fini de symboles (limitation : valeurs)
- de profondeur et d'arité non bornées

Plan

La grammaire, c'est quoi ?

Un langage—ensemble—d'arbres peut être spécifié par une grammaire (d'arbres)

$$
\begin{array}{rlll}
\mathcal{G} : & A \rightarrow & a\ [B, C?] \\
& B \rightarrow & b\ [C] \\
& C \rightarrow & c\ [D | (B, E^*)] \\
& D \rightarrow & d \\
& E \rightarrow & e
\end{array}
$$

- Le type A est le point d'entrée de $\mathcal{G}$
- Modèle de contenu entre crochets [.]
- $?\ +\ *\ |\ ,\ ()$ pour dessiner des expressions régulières

La grammaire, ça sert à quoi ?

Résolution de problèmes d'analyse statique
Quelques questions essentielles. . .

- Satisfiabilité : Requête q — Est-ce que $q(t) \neq \emptyset$ pour un certain t ?
- Inclusion : Requêtes q_1 et q_2 — Est-ce que $\forall t, q_1(t) \subseteq q_2(t)$?
- Équivalence : Requêtes q_1 et q_2 — Est-ce que $\forall t, q_1(t) = q_2(t)$?
- Intersection : Requêtes q_1 et q_2 — Est-ce que $\forall t, q_1(t) \cap q_2(t) = \emptyset$?

[diapos de Pierre Genevès, EPFL]

La grammaire, ça sert à quoi ? (bis)

- Expressivité : Comment exprimer une contrainte structurelle X avec un langage de définition de type Y ?
- Inclusion de types : Types d_1 et d_2 — Est-ce que $t \models d_1$ implique $t \models d_2$?
  Est-il possible de vérifier la compatibilité arrière lorsque mon type XML évolue ?
  Est-il possible de vérifier l'inclusion de types ?
- Contrôle de type : Types d_1 et d_2, Transformation T — Est-ce que $t \models d_1$ implique $T(t) \models d_2$?
  Peut-on être certain qu'une transformation XSL ne produira jamais de document invalide ?
- Inférence : Type d, Transformation T — Quel est le type de $T(t) \mid t \models d$?

Les classes de langages d'arbres

- Grammaire d'arbres locale : **DTD**
  $\rightsquigarrow$ *à une étiquette n'est associé qu'un seul type*
  Classe non close par composition $\{\cup\ \overline{X}\}$
- Grammaire d'arbres à types uniques : **XML Schema**
  $\rightsquigarrow$ à une étiquette n'est associé qu'un seul type *sous le même parent*
  Classe non close pour l'union
- Grammaire d'arbres régulière : **RelaxNG**
  $\rightsquigarrow$ récursivité sur les étiquettes en queue uniquement
  Classe close par composition, tests décidables (inclusion, etc.)
- Grammaire d'arbres hors-contexte : $a^n b^n$ $\emptyset$

Une grammaire régulière

```
Vente      = vente   [neuf[Neuf+] | occasion[Occasion*]]
Occasion   = véhicule[Modèle, Année]
Modèle     = modèle  []
Année      = année   []
Neuf       = PrixPublic+ | PrixCassé*
PrixPublic = véhicule[Modèle]
PrixCassé  = véhicule[Modèle, Remise]
Remise     = remise  []
```

Elle n'est pas à types uniques :
2 types étiquetés `véhicule` de modèle de contenu différent sous le même parent (`neuf`)

Quelques propriétés théoriques et pratiques

- $\mathcal{L}_{\text{local}} \subset \mathcal{L}_{\text{type unique}} \subset \mathcal{L}_{\text{régulier}} \subset \mathcal{L}_{\text{hors-contexte}}$
- Propriétés indécidables des grammaires hors-contexte :
  - L'inclusion
  - Le test de régularité
- Les grammaires d'arbre régulières autorisent la validation par événement (SAX)
- Une DTD requiert que les expressions régulières soient *déterministes* :
  $(b, c) \mid (b, d)$ doit être réécrit en $b, (c \mid d)$
- La vérification de type s'effectue en temps linéaire pour les grammaires d'arbre locales

Les automates d'arbres

Quadruplet (Q, I, F, Δ) sur l'alphabet Σ
Q : états, $I \subseteq Q$: é. initiaux, $F \subseteq Q$: é. finals, Δ : transitions
Transitions : $(q_1, \ldots, q_k) \rightarrow^{\sigma^{(k)}} q$ ou $q \rightarrow^{\sigma^{(k)}} (q_1, \ldots, q_k)$
Propriétés :

- On peut toujours *déterminiser* un automate d'arbre *ascendant*
- Les automates *descendants déterministes* ne reconnaissent pas tous les langages d'arbres réguliers
- Un langage d'arbres est régulier **ssi** il est reconnu par un automate d'arbres non-déterministe

Avantages : clôture / opérations décidables / outil théorique + algorithmique
Limitations : $a^n b^n$

XQuery

Langage de requêtes pour données semi-structurées

- Une requête XQuery est une **composition d'expressions**
- Chaque expression retourne une **séquence** (de nœuds et/ou de valeurs atomiques) ou une erreur, sans effet de bord
- Expressions (requêtes) simples :
  - valeurs atomiques : `46`,`"Salut"`
  - valeurs construites : `true()`, `date("2007-11-15")`
- Expressions complexes :
  - expressions de chemins (XPath 2.0)
  - expressions « **FLWR** » (`for-let-where-return`)
  - tests (`if-then-return-else-return`)
  - fonctions : racines, fonctions prédéfinies (XQuery 1.0 et XPath 2.0), fonctions utilisateur
- Opérateurs arithmétiques, booléens et de séquences

[diapos de Bernd Amann, ENST]

Requêtes XQuery

`Q0:document("bib.xml")/bib/book[1]/(editor union author)`
La requête retourne une séquence d'éléments de type `(editeur|auteur)` qui sont des enfants du premier élément de type `book` dans le document `bib.xml`

```
Q1: element{ document("bib.xml")//book[1]/name(@*[1]) } {
    attribute{ document("bib.xml")//book[1]/name(*[3]) } {
        document("bib.xml")//book[1]/*[3] }

Q2: <livres>
    { for $b in document("bib.xml")//book
        where $b/author/la="Rigaux" return
            if ($b/@year>2000)
            then <livre recent="true">{$b/@title}</livre>
            else <livre>{$b/@title}</livre>}
    </livres>

Q3: for $b in document("bib.xml")//book
    return element livre{
        attribute titre{$b/@title},
        for $a in $b/author
        return element auteur{
            attribute nom{$a/la},
            for $p in document("addr.xml")//person
            where $a/la=$p/name
            return attribute affiliation{$p/institution}}}
```

XQuery Update

W3C Working Draft 3 June 2005

- Classification des expressions XQuery selon :
  1. expression de mise à jour (*updating exp.*)
  2. expression en lecture seule (*non-updating exp.*)

  XQuery Update introduit **5 nouveaux types d'expressions** :
  - `insert`, `delete`, `replace`, `rename` : classe 1
  - `transform` : classe 2
- XQuery Update définit :
  - la catégorisation de toutes les expressions XQuery selon 1 ou 2
  - les contexte dans lesquels chaque catégorie peut apparaître
  - la **syntaxe** et la **sémantique** pour les nouvelles expressions

[diapos de Ioana Manolescu, INRIA]

Relationnel → XML

Comment présenter une BD relationnelle en XML ?

- Représentation *auto-descriptive* des tables :
  relation, n-uplet, attribut → élément (ou attribut ?)

```
<!ELEMENT Instruments (Instrument*) >
<!ELEMENT Intrument (Nom, Catégorie, Tonalité?) >
<!ELEMENT Nom (#PCDATA) >
<!ELEMENT Catégorie (#PCDATA) >
<!ELEMENT Tonalité (#PCDATA) >
```

- Modèle générique
  les éléments sont `Relation` `Tuple` `Attribut`...
- Avec SQL:1999 (SQL-3) :
  relations imbriquées pour plus de structuration
- Avec SQL:2003 : extension SQL/XML

```
SELECT XMLELEMENT(NAME "Instrument", XMLATTRIBUTES(Catégorie), Nom) AS élément
FROM Instruments;

Résultat : <Instrument Catégorie="cuivre">trompette</Instrument>
```

Stockage de documents XML

- Propriétés des documents XML
  - Identité : 2 éléments à contenu identique sont différents
  - Ordre des éléments
  - Structuration
  - Contenu mixte
  - Variabilité structurelle : `,*+?()`
  - Schéma ?
- Solutions de stockage :
  - Sérialisation dans des fichiers « à plat »ou champs CLOB
  - Bases de données relationnelles (*Mapping*)
  - Bases de données relationnelles étendues (TAD XML)
  - Bases de données XML natives (modèle physique différencié)
- Modèle de données : **flot**, **n-uplet** ou **arbre**
- Compromis fondamental : degré de **fragmentation**

Appariement XML-Relationnel

Schéma de stockage...

- indépendant de la DTD/du schéma XML
  - représentation des *arêtes* (ou des nœuds)
  - représentation des chemins
- guidé par la DTD/le schéma XML
  - uniquement le schéma :
    basic/shared/hybrid Inlining à l'aide d'un **graphe structurel**
  - schéma + informations de coût (*workload*) :
    idem + règles de transformation de schéma
- défini par l'utilisateur
  correspondance manuelle XML $\leftrightarrow$ Relationnel/Objet

Liste d'adjacences

- Ordre préfixe pour la numérotation des nœuds
  `EdgeTable(nodeId, parent, tag, val)`

Optimisé en
`EdgeTable(nodeId, parent, pathId, val)` et
`PathTable(pathId, path)`

- Les chemins racine-élément identiques sont représentés une seule fois dans `PathTable`

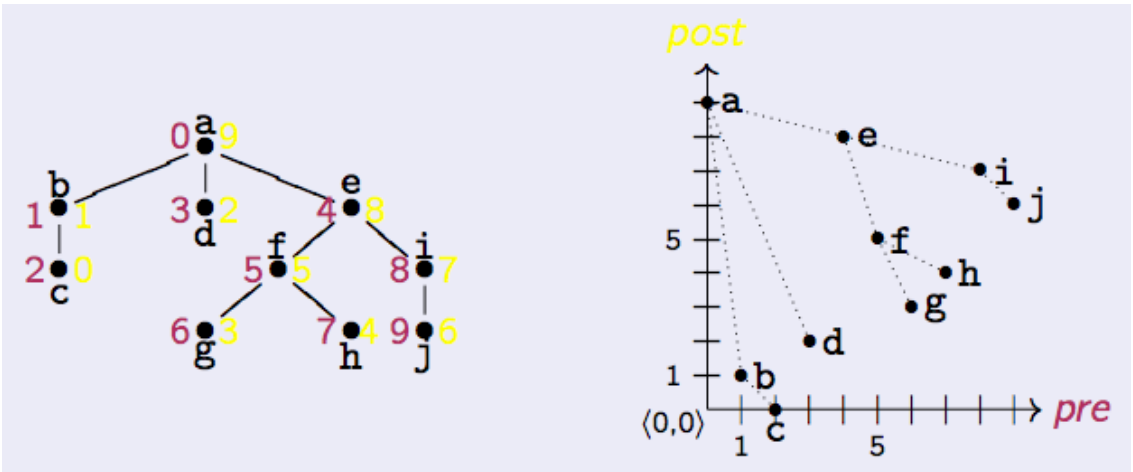
Critiques :

- requêtes XPath au-delà de FO (récursivité)
- relation parent-enfant, à l'exclusion des autres axes XPath

Ensembles emboîtés

Objectif : optimiser les requêtes XPath

- **identifiant structurel** : [pré-ordre, post-ordre, profondeur]



  - Schéma de Dietz, 87
  - Relations structurelles / axes XPath :
    (n_1 ancêtre de n_2) $\Leftrightarrow (n_1.\text{pre} < n_2.\text{pre}) \wedge (n_1.\text{post} > n_2.\text{post})$
- Table `accel(pre, post, parent/level, tag, text)`
  - Indexation par arbre B+ (plaçant sur pré, non plaçant sur post)
  - Intuition spatiales : indexation par arbre R

Inlining (1/2)

- Point de départ :
  - simplifier la DTD : $*$ seulement
  - 1 élément $=$ 1 relation, 1 attribut $=$ 1 colonne
  - clés étrangères pour les liens
  - sujet à la fragmentation outrancière
- Principe de l'*Inlining* : autant de descendants que possible dans la même relation
  - sujet à la redondance (chaque élément est une racine locale)
  - nécessité de détecter/casser la récursivité

Inlining (2/2)

- Technique *Shared Inlining*
  - nouvelle relation pour éléments partagés, récursifs ou multiples
  - 1 seule représentation par nœud du document
  - CNS : degré entrant ≥ 2 dans le graphe de DTD simplifié
  - Réduction du nombre de requêtes imbriquées pour des expressions de chemin
- Technique *Hybrid Inlining*
  - idem, sauf pour éléments partagés (qui sont donc embarqués)
  - Elimination des jointures pour les éléments partagés

NXD ou le SGBD XML natif

Spécifiquement conçu pour les données semi-structurées (XML)

- Entité logique : document XML (modèle d'arbre ordonné)
- Modèles très simples de stockage d'arbres
- **Navigation sur disque** dans un graphe persistant (OEM)
- Principe de l'**indexation de graphe** :
  1. partitionner les nœuds en **classes d'équivalence** (invariants, régularités, etc.)
  2. stocker les nœuds selon les classes d'équivalence
  3. utiliser les classes comme réponses pré-calculées aux requêtes
- Multiples techniques d'indexation de graphe
  - Indexation d'objets dans un graphe
  - Indexation de chemins (DataGuides, F&B index, etc.)
  - Simplification de document : les schémas de graphes

[diapos de Ioana Manolescu, INRIA]

NXD du marché (1/2)

État de l'art partiel et partial

- Mark Logic
  - `http://www.marklogic.com/`
  - « Oracle des bases de données XML natives »
  - Opérations CRUD, volumétrie supportée, fiabilité...
  - Commercial, licence coûteuse
- xDB (X-Hive)
  - Acheté par EMC (Documentum)
  - Orienté gestion de documents, volumétrie et performance
  - Commercial
- Berkeley DB XML
  - `http://www.oracle.com/database/berkeley-db/xml/index.html`
  - Projet Open Source développé par Sleepycat, repris par Oracle
  - Fiable et mature
  - Conception originale : bibliothèque embarquée (C, Java,...)

NXD du marché (2/2)

État de l'art (suite)

- eXist
  - `http://exist.sourceforge.net/`
  - Référence des bases de données XML natives Open Source
  - En constante amélioration
  - Comparé à MySQL
- MonetDB/XQuery
  - `http://monetdb.cwi.nl/XQuery/index.html`
  - Projet Open Source développé au CWI
  - Repose sur *XPath Accelerator*, *StairCase Join*, *Loop Lifting*
- BaseX
  - `http://www.inf.uni-konstanz.de/dbis/basex/index`
  - Vitrine du groupe de recherche DBIS, Konstanz
  - Léger, performant

[diapos de François Goldgewicht, Valtech]

Quelques références

- Site du W3C `http://www.w3c.org/`
- S. Abiteboul, P. Buneman, D. Suciu (1999). *Data on the Web : From Relations to Semistructured Data and Xml*, Morgan Kaufmann.
- B. Amann et P. Rigaux (2002). *Comprendre XSLT*, O'Reilly.
- K. Williams, M. Brundage, P. Dengler et al. (2001). *XML et les bases de données*, Eyrolles.
- J.-J. Thomasso (2003). *Schémas XML*, Eyrolles.
- G. GARDARIN (2002). *XML : des bases de données aux services Web*, Dunod.
- H. Comon, M. Dauchet, R. Gilleron et al. (2007) Tree Automata Techniques and Applications. Disponible en ligne `http://tata.gforge.inria.fr/`