Processeur XPath

ESIAL 2011 - Mini-projet de SD

Julien VAUBOURG && Tristan STEF

22 mai 2011

Table des matières

1 Conception

1.1 Une grammaire variable

Plutôt de nous contraindre à la grammaire de la DTD proposée, nous avons essayé d'imaginer une solution pour que le projet puisse être réutilisé avec d'autres grammaires, sans pour autant écrire un *parseur* de DTD.

Pour ce faire, nous avons décidé d'utiliser le concept de premiers et de suivants, vu en *Mathématiques pour l'Informatique 2*. Ainsi, chacune des balises XML faisant partie des terminaux de la grammaire, peut indiquer quels sont les terminaux qu'elle peut accepter en premier, et quels sont ceux qui peuvent la suivre directement.

De cette façon, en partant d'un terminal précis que nous avons appelé *l'axiome* (indiqué en haut de *Main.java*), il est possible de connaitre toute la grammaire sous-jacente. Des terminaux internes ont été ajoutés : *text*, qui indique que la balise peut contenir du texte (tout comme les interpréteurs DOM considèrent ce noeud implicite), et *void* qui sert à préciser qu'il peut ne rien y avoir à la suite de la balise (potentielle dernière de son parent, donc).

Actuellement, ces informations sont indiquées dans des classes presque vides, du *package* dédié *Grammaire*. Le nom de la classe indique directement le nom du terminal. Une autre solution aurait été de ranger ces informations dans un fichier

texte, consulté au lancement de l'application, qui aurait permis de changer de grammaire sans recompiler l'application. C'est une évolution qu'il serait aisé de faire (mais pas nécessaire dans le cadre de ce projet).

1.2 Deux analyseurs et un interpréteur

Nous avons tenté autant que faire se peut d'attribuer leur vrai rôle à chacun des deux analyseurs. Il s'exécutent séquentiellement.

1.2.1 L'analyseur lexical

Il récupère les caractères du fichier XML un par un, et se prononce dès qu'il repère un terminal de la grammaire. Il précise alors s'il s'agit d'une balise ouvrante, fermante, ou de texte.

Il est apte à détecter un certain nombre d'erreur, qu'il ne manquera pas de renvoyer sous forme d'exceptions lexicales précises.

Il vérifie ainsi :

- La syntaxe du XML (exemple : `<téléphone !>`).
- L'existence du terminal dans la grammaire (exemple : `<anniversaire>`).
- Que les attributs trouvés sont bien acceptés par la balise porteuse (exemple : `<personne type="bleu">`).
- L'unicité des attributs trouvés pour une même balise (exemple : `<telephone type="gsm" type="fixe">`).

1.2.2 L'analyseur syntaxique

Dès que l'analyseur lexical déclare qu'il a trouvé un terminal, celui-ci est immédiatement transmis à l'analyseur syntaxique.

Ce dernier se charge de créer la représentation mémoire de l'arbre XML. Chacune des classes correspondantes à une balise contiendra ainsi la liste de ses fils directs. En parcourant les fils depuis l'axiome, il est possible de reparcourir toute l'arborescence.

Durant cette construction, il se charge de vérifier ce qui est spécifique à son rôle et est apte à renvoyer une exception syntaxique concernant :

- L'ordre des balises (exemple :
  `<naissance><jour>...</jour><annee>...</annee><mois>...</mois></naissance>`).
- La possibilité ou non d'avoir du texte dans une balise (exemple :
  `<personne>du texte (...)</personne>`).
- Le minimum qu'une balise doit posséder (exemple :
  `<naissance><mois>...</mois><annee>...</annee></naissance>`).
- La bonne fermeture des balises (exemple :
  `<naissance><jour>...</jour><mois>...</mois><annee>...</naissance>`).

1.2.3 L'interpréteur XPath

Il interprète les requêtes XPath, et retourne la liste des noeuds qui correspondent, s'ils existent.

Il est capable de repérer les requêtes qui ne respectent pas le format requis et d'indiquer précisément où se situe le problème.

Nous avons choisi d'utiliser des expressions régulières capturantes efficaces pour décomposer la requête, et de la traiter récursivement en réduisant chaque fois la requête d'un niveau de noeud et en passant chaque fois les noeuds qui correspondent au début de la requête. Lorsque la requête est vide ou qu'il n'y a plus de résultats correspondants, l'interprétation s'achève.

1.3 Options implantées

1.3.1 Attributs

Les attributs sont acceptés dans les balises, et sont renseignés par les terminaux en même temps que leurs premiers et leurs suivants.

Nous avons choisi de les rendre facultatifs et de type *CDATA* par défaut.

Il est ainsi possible de filtrer par argument (voir *Prédicats* ci-dessous) ou de demander le contenu d'un attribut :

- `//personne/telephone/@type` :
  Retourne la valeur de l'attribut *type* des téléphones des personnes, lorsqu'ils sont précisés.

Le nombre d'argument est illimité.

1.3.2 Prédicats

Des prédicats sont utilisables dans les requêtes XPath. Ils peuvent utiliser les arguments et peuvent se cumuler d'un noeud à l'autre.

Exemples :

- `/annuaire/personne[2]/telephone` :
  Retourne la balise *telephone* de la seconde personne trouvée dans l'annuaire.
- `//personne[naissance]/telephone` :
  Retourne la balise *telephone* des personnes ayant indiqué leur date de naissance.
- `//personne/telephone[@type]` :
  Retourne les balises *telephone* des personnes qui ont indiqué le type du téléphone.
- `//personne/telephone[@type='gsm']` :
  Retourne les balises *telephone* des personnes ayant indiqué des téléphones de type GSM.

1.3.3 Méthode *text()*

Étant donné notre noeud *text* qui compose chacun des noeuds textes, il a été aisé d'implémenter la possibilité de ne se faire renvoyer que le contenu d'une balise.

- `//personne[naissance]/telephone[@type]/text()` :
  Retourne directement les numéros de téléphones filtrés, sans leurs balises *<telephone>* et *</telephone.*

2 Tests unitaires

2.1 L'analyseur lexical

Différentes classes d'équivalence ont été définies :

- Cas d'une balise
- Cas d'une balise suivie de texte
- Cas d'une balise suivie d'une autre balise
- Cas d'une balise suivie d'une balise malformée
- Cas d'une balise suivie d'une balise ouvrante inconnue
- Cas d'une balise suivie d'une balise fermante inconnue

Afin d'être exhaustif, ce schéma pour « une balise » a été suivi pour chacune de ces entités :

- Balise conforme
- Texte
- Balise malformée
- Balise ouvrante inconnue
- Balise fermante inconnue

Chaque fois, le nom du terminal trouvé, ainsi que son type et les caractères qui ont été nécessaires pour le déterminer ont été vérifier par des assertions.

Des tests spécifiques ont été réalisés concernant les balises avec attributs.

Un certain nombre de tests sont chargés de vérifier que les exceptions attendues sont bien retournées dans les différents cas d'erreur possibles.

Ainsi, *AnalyseurLexicalTest* contient pas moins de 102 assertions à lui tout seul.

2.2 L'analyseur syntaxique

Différentes classes d'équivalence ont été définies :

- Cas d'un arbre encore vide
- Cas d'un arbre qui ne contient que la racine (l'axiome)
- Cas d'un arbre qui contient plus que la racine

Pour chacune de ces classes, nous avons testé :

- Ajout d'une première balise fille compatible
- Ajout d'une première balise fille incompatible
- Ajout d'une seconde balise fille, incompatible (suivante impossible de la première)
- Ajout d'une seconde balise fille, compatible

Pour la méthode spécifique à la fermeture des balises :

- Cas d'un arbre qui ne contient que la racine
- Cas d'une arbre qui contient plus que la racine

Pour chacun de ces cas, nous avons testé :

- Fermeture de la racine seule
- Fermeture d'un premier incomplet
- Fermeture d'un premier complet
- Fermeture d'un suivant incomplet
- Fermeture d'un suivant complet

Nous avons chaque fois testé le retour de la commande *xml()* pour vérifier que les noeuds contenaient bien ce que nous souhaitions.

Des tests spécifiques aux attributs ont été réalisés, ainsi que des tests de retour des erreurs provoquées.

2.3 L'interpréteur XPath

Un certain nombre de tests ont été réalisés.

Requêtes commençant par un oblique (chemin absolu) :

- Avec une seule balise (Annuaire)
- Avec plus d'un niveau de balises
- Ne retournant aucun résultat
- Commençant pas par la balise racine
- Contenant une balise inconnue
- Ne respectant pas l'ordre des balises
- Se terminant mal
- Vide

Requêtes commençant ou contenant un double oblique (chemin relatif) :

- Directement une feuille
- Double oblique au milieu
- Vide

Tests des prédicats :

- Numérique (final)
- Numérique (au milieu)
- Numérique hors index
- Testant l'existence d'un noeud
- Plusieurs prédicats
- Caractères non attendus

Tests des attributs :

- Valeur d'un attribut
- Avec prédicat sur l'attribut

– Avec prédicat sur le contenu de l'attribut
– Attribut inexistant

Tests de la fonction `text()` :

– Avec un noeud texte
– Avec un noeud non-texte

3 Schéma relationnel des classes