Rapport du projet XMLLiteParser

Modélisation

Mathis Deloge, Antoine Petot, Ange Picard

1 Descriptif du sujet

Comme cité dans le sujet, un parseur / validateur XML-Lite est un programme capable de lire un fichier, d'indiquer s'il vérifie la norme XML-Lite et si oui, de l'analyser et de retenir sa structure ainsi que son contenu. Pour nous permettre de concevoir un programme réalisable, notre parseur / validateur opère sur un langage simplifié de XML, le XML-Lite conçu pour faciliter l'utilisation, les performances ainsi que les normes de conformité (XML 1.0).

1.1 Le XML-Lite

Pour être considéré comme du XML-Lite, les fichiers parsés / validés par notre programme doivent respecter certaines règles :

- Une balise possède un nom.
- Une balise doit être ouverte puis fermée.
- Une balise peut contenir du texte.
- Une balise peut contenir d'autre balises.
- L'ordre des balises filles n'a pas d'importance et tout le texte contenu dans une balise est regroupé en un seul bloc.
- Une balise fille doit être fermée avant la fermeture de la balise parent.
- Une balise peut contenir une balise du même nom.
- Un document doit commencer par l'ouverture d'une balise se fermant à la fin du document.

1.2 Structure du document

Le parseur / validateur doit être capable de lire n'importe quel fichier XML-Lite mais doit aussi être en mesure d'attendre une certaine structure de document grâce à l'ajout d'un fichier .dtd appelé schéma. Grâce aux fichiers schéma, le parseur / validateur connais avec plus de finesse les balises filles autorisées ou non pour chaque balises. C'est une sorte de modèle qui permettra la validation du fichier XML-Lite.

2 Journal de bord

2.1 Séance 1

Lors de la première séance, nous avons tout d'abord effectué le choix de sujet. Le parseur / validateur XMLLite nous a intéressé étant donné le grand nombre de programmes fonctionnant avec XML pour la persistance et la souplesse de ce format de base de données, nous étions intéressé de découvrir les notions de bases du XML.

Par ailleurs, durant cette séance, nous avons trouvé des informations sur les validateurs de documents et avons pensé à implémenter un automate fini pour modéliser notre validateur. Le design objet "state pattern" semblait particulièrement adapté.

2.2 Séance 2

Lors de la deuxième séance, nous avons modélisé l'automate fini schématiquement, puis, nous l'avons implémenté. Il est utilisé pour valider le document. Nous avons également codé tous les différents états.

Exemple d'un état du validateur

```
1 public class NewTag implements State {
2     @Override
3     public State transition(char c) {
4         if (c == '/')
5             return new NewClosingTag();
```

```
6           else if ((c != '<') && (c != '>')) {
7               XMLLiteParser.getInstance().fillBuffer(c);
8               return new NewTagName();
9           } else
10              return new Error();
11      }
12
13      @Override
14      public boolean isFinal() {
15          return false;
16      }
17  }
```

Puis, nous avons réfléchis à la structure mathématique du pasreur, nous sommes vite arrivé à celle d'un arbre. Cette structure à l'avantage d'être facile à designer en objet. Nous avons donc implémenté deux classes :

- XMLLiteNode : Pour représenter une feuille ou un nœud de l'arbre.
- XMLLiteParser : Pour construire l'arbre.

Il a également fallu implémenter un buffer afin de stocker caractère par caractère les informations provenant des états du validateur.

3 Choix du modèle mathématique

3.1 Le modèle mathématique

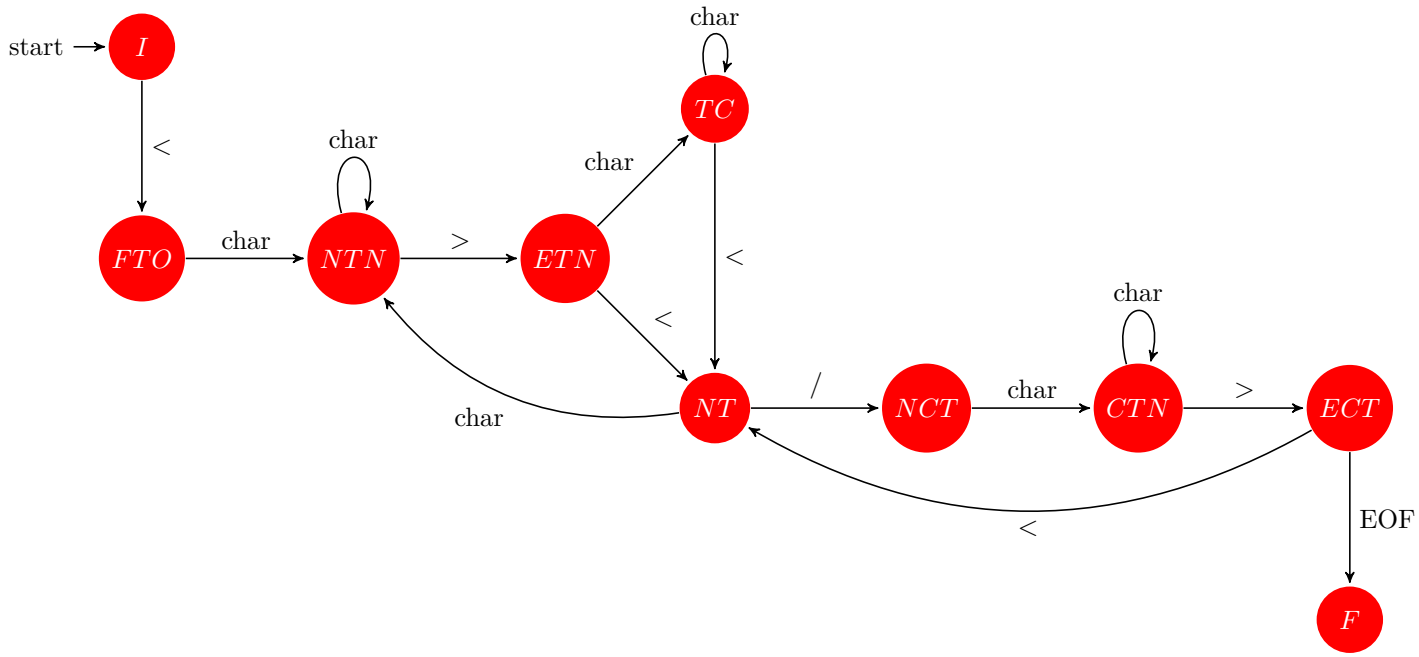
Pour nous permettre de parcourir rapidement un fichier XML-Lite, nous avons opté pour le développement d'un automate fini.

Tout d'abord puisque grâce à la simplicité et la rigidité du langage XML-Lite, il y a très peu d'états différents lors de la lecture d'un fichier. Les transitions entre états se font uniquement grâce à la différenciation des caractères '<', '>', '/' et le reste.

Cette façon de parcourir un fichier XML-Lite caractère par caractère s'est avérée très rapide (exécution en 13ms pour un fichier XML-Lite de près de 700Mo).

3.2 Représentation de l'automate finis

3.2.1 Schéma



3.2.2 Description des états

I Initial

FTO First Tag Opening

NTN New Tag Name

ETN End Tag Name

TC Text Content

NT New Tag

NCT New Closing Tag

CTN Colsing Tag Name

ECT End Closing Tag

F Final

4 Prolongements possibles

4.1 Étudiez et justifiez les propriétés de la structure mathématique utilisée.

4.2 Modifiez votre validateur afin qu'il permette le débogage du fichier XML. Quel impact cette modification a eu sur la structure mathématique utilisée ?

4.3 Modifiez votre validateur afin qu'il s'accorde à un schéma prédéfini. Quel impact cette modification a eu sur la structure mathématique utilisée ?

4.4 Modifiez votre validateur afin qu'il prenne en compte un schéma accompagnant éventuellement un fichier XML-Lite.

4.5 Proposez un schéma permettant de stocker la base de données d'un générateur de QCM, chaque question ayant de 1 à 5 réponses, correctes ou non.

4.6 Rajoutez à votre programme un interpréteur (pour le schéma du prolongement précédent).

5 Conclusion

6 Comment ajouter du code ?

6.1 Comme ça

```
1  import java.io.IOException;
2  import java.util.Date;
3  import java.util.Timer;
4
5  /**
6   * Created by MrMan on 12/09/2016.
7   */
8  public class main {
9      public static void main(String[] args) {
10         TransitionSystem ts = new TransitionSystem();
11         try {
12             ts.openXMLFile("XMLDocs\\Success.xmll");
13         } catch (IOException e) {
14             e.printStackTrace();
15         }
16         long startTime = System.currentTimeMillis();
17         long elapsedTime;
18         ts.start();
19         elapsedTime = (new Date()).getTime() - startTime;
20         System.out.println("Document validated in " + (elapsedTime) + " ms");
21         TreeView tv = new TreeView();
22         tv.setVisible(true);
23     }
24 }
```

6.2 Ou comme ça

```
1 class HelloWorldApp {
2     public static void main(String[] args) {
3         System.out.println("Hello World!"); // Display the string.
4         for (int i = 0; i < 100; ++i) {
5             System.out.println(i);
6         }
7     }
8 }
```