République Algérienne Démocratique et Populaire
Ministère De L'enseignement Supérieur Et De La Recherche Scientifique
Université Amar Télidji Laghouat

Faculté Des Sciences

Département D'informatique

Mémoire en vue de l'obtention du diplôme de Master en informatique

Thème :

Langages d'arbres rangés et non-rangés pour la modélisation des documents XML

Présenté par :
BADAOUI Mehdi Bachir

Composition du jury :

Mr. Y.Ouinten	M C (A)	université de Laghouat	Président
Mr. T.BENDOUMA	M A (B)	université de Laghouat	Examinateur
Mr.	M A (A)	université de Laghouat	Examinateur
MM. H.CHERROUN	M C (A)	université de Laghouat	Rapportrice
M. Y.GUELLOUMA	M A (A)	université de Laghouat	Co-Rapporteur

2010 / 2011

Remerciement

Je suis doublement content d'écrire cette partie : d'abord, le Master est déjà un pas franchi et ensuite, elle ne sera pas révisée par mes encadreurs.

Je remercie Dieu qui m'a permis de réussir dans mes études, Mes parents qui m'ont toujours soutenu tout au long de mon Cursus ainsi que tous les moments difficile par lesquels je suis passé.

Il y a de nombreuses personnes que je voudrais remercier pour le soutien durant toutes mes années d'études.

Tout d'abord je voudrais remercier très sincèrement les membres du jury : MM. H.CHERROUN d'avoir acceptée d'être rapporteure de mes mémoires de Licence et Master. C'est toujours un plaisir de travailler avec elle en espérant qu'un jour j'écrirai mieux qu'elle des articles scientifiques.

Mr. Y.GUELLOUMA d'avoir acceptée d'être mon Co-Rapporteur de mes mémoires de Licence et Master. Et pour tous les conseils qu'il m'a donnés durant mes Cinq années d'études.et je me rappellerai toujours des moments qu'on a passé ensemble, en espérant qu'un jour je serai un meilleur programmeur que lui.

Mr. Y. Ouinten m'a fait l'honneur d'accepter de présider ce jury

Les examinaterus d'avoir accepté d'examiner mon travail et de me faire honneur de leurs présences au sein du jury.

Je serai toujours reconnaissant à tous mes enseignants qui n'ont jamais hésité à nous faire passer leurs informations.

Dédicace

A mes parents, A mes frères et sœur Amine, Mustapha et Fazo.
Vous vous êtes dépensés pour moi sans compter.
En reconnaissance de tous les sacrifices consentis par tous et chacun pour me permettre d'atteindre cette étape de ma vie. Avec toute ma tendresse.

A mon futur neveux.
Meilleurs vœux de succès dans tes études.

A mes Grands-parents, oncles, tantes, cousins et cousines.
Vous avez de près ou de loin contribué à ma formation.
Affectueuse reconnaissance

A mes camarades de classes et tous ceux du département Informatique de l'Université Amar TELIDJI.

A mes collègues de Elite SCHOOL
Merci de m'avoir accepté au sein de votre équipe malgré mon jeune âge.
Respectueuse reconnaissance

A mes Amis du Club AQUA
Je commence par mon petit groupe de nageur avec qui j'espère avec beaucoup de réussite, les nageurs avec qui j'ai eu l'occasion de nager durant ses six dernières années, ainsi que tous les entraîneurs et personnel de ce club.

A mes plus proche amis
youcef, Djamel,Sadek et Abdellah pour leurs soutiens et leurs épaulements dans ma vie de tout les jours.

Et une dernière dédicace aux personnes qui me sont chères et que j'ai perdu, je pense surtout à mon oncle Naser, mon grand-oncle ABDELLAH et sa femme.

Table des matières

Table des figures

Introduction générale

Ces dernières Années le monde a connu de très grands changements notamment dans la vie quotidienne de chaque individu, l'importante avancée technologique a bouleversé le monde et Internet en est une, elle a non-seulement changée nos vies mais elle a participé en grande partie à l'évolution des autres domaines technologiques.

Internet est devenu la source d'information numéro un au monde tout y ait, n'importe quel personne possédant de l'information peut la mettre sur la toile, et vu ce grand flux d'informations qui se trouve et le problème de la recherche qui s'impose, il est devenu essentiel de structurer ces informations et de les organisées pour faciliter la recherche aux internautes car pour un tel sucées Internet doit être plus rapide que jamais.

Tout le monde à la possibilité de mettre ses informations sur le Net soit en privé (dans les boites mail) soit en public (dans les sites, les blogs...), alors un autre problème se pose celui du grand volume de données qu'Internet supporte et devra supporter.

L'informatique fondamentale offre un large spectre de modèles pour gérer, reconnaitre et stocker l'information. En effet, les automates et langages classiques ainsi que les automates d'arbre et leurs techniques représentent de puissants outils et méthodes.

L'objectif de notre travail est de présenter, étudier ces outils de la théories des arbres en vu de les instrumenter pour la modélisation des documents XML, on traitera particulièrement les langages d'arbres rangés et non-rangés ainsi que leurs automates.

Notre memoire est scindé en quatre chapitres, dans le premier chapitre on introduit les concepts généraux du langage XML, dans le deuxième chapitre on commence par les automates sur les mots, en enchaine avec les auto-

mates d'arbres, dans le troisième chapitre on expose les problèmes liés à la modélisation des documents XML en utilisant les automates d'arbres tout on introduisant deux nouveaux types d'automate d'arbre particuliers. Finalement on présente un outil de définition d'automates pour les documents XML.

Chapitre 1

Documents XML et traitements associés

Introduction

Le flux de données existant sur le net est tellement important qu'il doit être structuré pour faciliter sa lecture, son stockage et notamment sa compression. Le langage XML permet cela.
Depuis sa première publication, en 1998, le langage XML est devenu un format de balisage des données utilisées par de nombreuses applications. Cela s'explique par le fait que XML est capable de baliser aussi bien un contenu Web que des données utilisées par les applications [5].

Grâce aux balises, l'ordinateur est également capable d'en traiter le contenu et de bien séparer les informations. C'est un des avantages du XML : c'est l'un des rares formats qui peut être à la fois lu par un humain et par un ordinateur. De plus c'est un format interopérable. En effet, si l'on se met d'accord sur les balises à utiliser, on peut utiliser XML pour échanger des informations entre différentes personnes et logiciels. XML, grâce à l'utilisation de l'encodage d'UTF-8, supporte très bien tous les alphabets du monde [13].

1.1 XML

L'acronyme de " Extensible Markup Language " en français " langage extensible de balisage ", est un langage informatique de balisage générique. Il sert essentiellement à stocker et transférer des données de type texte unicode structurées en champs arborescents. Ce langage est qualifié d'extensible car il permet à l'utilisateur de définir les balises des éléments [3].

1.1.1 Exemple de document XML

```
< ? xml version= "1.0" encoding ="iso-8859-1" ?>(1)
<!-- Time-Stamp : "bibliography.xml 3 Mars 2008 16 :24 :04" -- >(2)
< ! DOCTYPE bibliography SYSTEM "bibliography.dtd" >(3)
<bibliography>(4)
    <book Key="Michard01" lang="Fr">
      <Title> XML langage et application </title>
      <author> Alain Michard </author>
      <year>2001</year>
      <publisher>Eyrolles</publisher>
      <isbn>2 - 212 - 09206 - 7</isbn>
      <url>http ://www.editions-eyrolles/livres/Michard/ </url>
    </book>
    <book Key="zeldman03" lang="en">
      <title>Designing with web standars </title>
      <author>Jeffrey zeldman </author>
      <year>2003</year>
      <publisher>New Riders</publisher>
      <isbn>0 - 7357 - 1201 - 8</isbn>
    </book> </bibliography>(5)
```

- (1) Entête XML avec la version 1.0 et l'encodage iso-8859-1 des caractères.
- (2) Commentaire délimité par les chaînes de caractères <!-- et -- >
- (3) Déclaration de DTD externe dans le fichier bibliography.dtd
- (4) Balise ouvrante de l'élément racine bibliography
- (5) Balise fermante de l'élément racine bibliography [3].

1.1.2 Représentation d'un document XML sous forme d'arbre

l'exemple précèdent sera représenté naturellement par l'arbre de la Figure 1.1

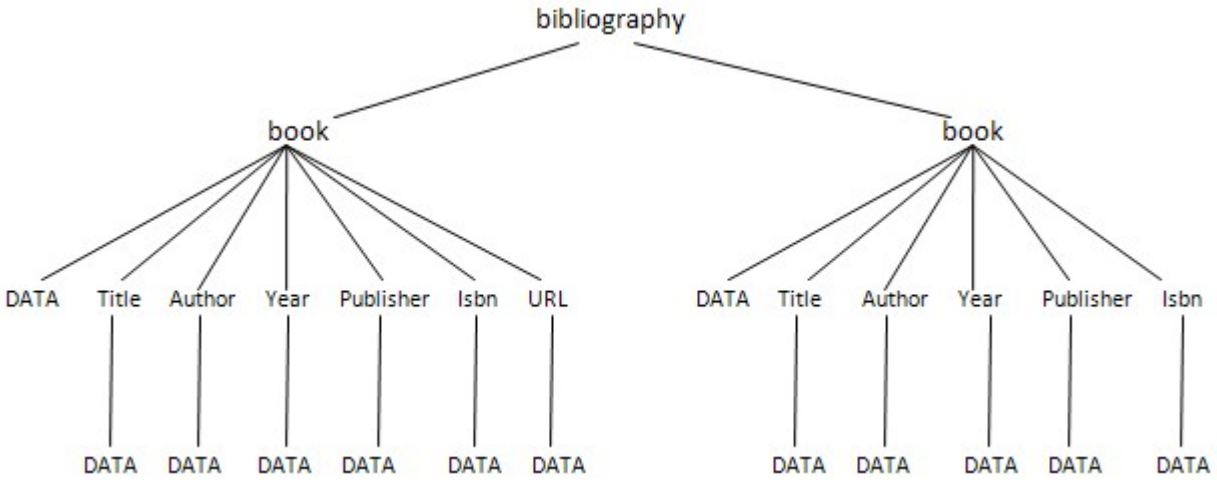

FIGURE 1.1 – Arbre du document XML

1.2 Les DTDs

En peu de temps XML est devenu le langage le plus utilisé sur le Web. En effet, presque tous les documents du Web sont structurées comme tel, vu sa flexibilité, chaque utilisateur peut ajouter des balises qui lui sont spécifique, cela exige une vérification de conformité du document. les DTDs (Les schémas XML ensuite) sont les responsables de cette vérification.

1.2.1 Exemple de DTD

On reprend la petite bibliographie du fichier bibliography.xml déjà utilisée précédemment. Le nom bibliography de l'élément racine du document apparaît dans cette déclaration juste après le mot clé DOCTYPE.

< !DOCTYPE bibliography SYSTEM "bibliography.dtd" >

On présente maintenant le contenu de ce fichier bibliography.dtd qui contient la DTD du fichier bibliography.xml. La syntaxe des DTD a été héritée du SGML [1] et elle est différente du reste du document XML.

```
<! ELEMENT bibliography (book)+ >(1)
<! ELEMENT book (title, author, year, publisher, isbn, url ?) >(2)
<! ATTLIST   book Key NMTOKEN # REQUIRED >(3)
<! ATTLIST   book lang (Fr | en)      # REQUIRED >(4)
<! ELEMENT title                     (# PCDATA) >(5)
<! ELEMENT author                    (# PCDATA) >
<! ELEMENT year                      (# PCDATA) >
<! ELEMENT publisher                 (# PCDATA) >
<! ELEMENT isbn                      (# PCDATA) >
<! ELEMENT url                       (# PCDATA) >
```

- (1) Déclaration de l'élément bibliography devant contenir une suite non vide d'éléments book.
- (2) Déclaration de l'élément book devant contenir les éléments title, author, isbn et url.
- (3) et (4) Déclarations des attributs obligatoires key et lang de l'élément book.
- (5) Déclaration de l'élément title devant contenir uniquement du texte [3].

1.2.2 Représentation d'une DTD sous forme d'arbre

Même une DTD peut être représentée naturellement par un arbre.

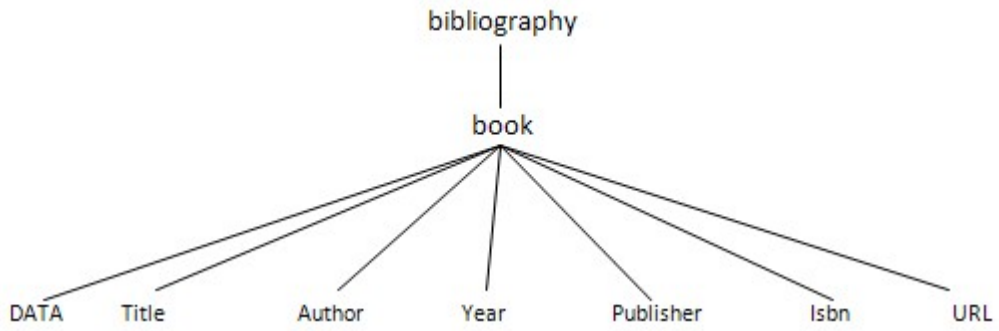

FIGURE 1.2 – Arbre d'une DTD

1. Standard Generalized Markup Language (langage normalisé de balisage généralisé - SGML) est un langage de description à balises, de norme ISO (ISO 8879 :1986)

1.2.3 Inconvénients des DTDs

L'usage des DTDs permet de vérifier la validité d'un document XML mais, elles possèdent des inconvénients

- Syntaxe non XML
  une syntaxe pour les documents XML, une autre pour les DTD → incohérence
- support limité des types de données
  Pas d'intervalle sur les entiers par exemple, pas d'extensibilité, d'héritage...
- Besoin d'un système de type plus riche, permettant de définir des schémas. En effet, une DTD supporte 10 types, cependant un schéma XML supporte plus de 44 types.

1.3 Les schémas XML

Les inconvénients des DTDs ont poussé les chercheurs à concevoir une nouvelle technique de vérification de validité des documents XML qui est le schéma XML.

Les Schémas XML sont une extension significative des DTDs :

- extensible (on peut définir ses propres types)
  même syntaxe que les documents XML : support de concepts objets

- définition d'un type par extension ou restriction d'un autre
  support des ensembles : notion de "clé"

1.3.1 Exemple de schéma Xml

```
< ? xml version= "1.0" encoding ="iso-8859-1" ?>
<xsd :schema xmlns :xsd "hhtp :/ /www.w3.org/2001/XMLSchema">(1)
   <xsd :annotation>(2)
      <xsd :documentation xml :lang="fr">
            Schéma XML pour bibliography.xml
      </xsd :documentation>
   </xsd :annotation>
   <xsd :element name="bibliography" type= "bibliography" />(3)
<xsd :complexeType name="bibliography" >(4)
   <xsd :sequence>
    <xsd :element name="book" minOccurs="1" maxOccurs="unbounded">(5)
```

```
      <xsd :complexeType>
       <xsd :sequence>
          <xsd :element name="title" type="xsd !string"/>
          <xsd :element name="author" type="xsd !string"/>
          <xsd :element name="year" type="xsd !string"/>
          <xsd :element name="publisher" type="xsd !string"/>
          <xsd :element name="isbn" type="xsd !string"/>
          <xsd :element name="url" type="xsd !string" minOccurs="0"/>
       </xsd :sequence>
       <xsd : attribute name="key" type="xsd :NMTOKEN" use="required"/>(6)
       <xsd : attribute name="lang" type="xsd :NMTOKEN" use="required"/>(7)
      </xsd :complexeType>
     <xsd :element>
    </xsd : Sequence>
   </xsd : complextype>
  </xsd : Sequence>
 </xsd : schema>
```

- (1) Élément racine xsd :schema avec la déclaration de l'espace de noms des schémas associé au préfixe xsd.
- (2) Documentation du schéma.
- (3) Déclaration de l'élément bibliography avec le type Bibliography.
- (4) Début de la définition du type Bibliography.
- (5) Déclaration de l'élément book dans le contenu du type Bibliography.
- (6) et (7) Déclaration des attributs key et lang de l'élément book avec le type xsd :NMTOKEN [3].

1.3.2 Représentation d'un schéma XML sous forme d'arbre

Pour les schéma XML, l'arbre permet de les representer naturellement ainsi si on prend l'exemple du schéma XML précèdent il aura comme représentation l'arbre suivant

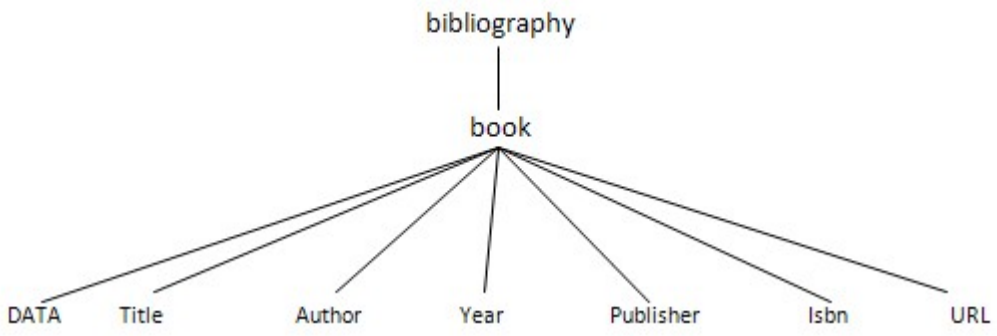

FIGURE 1.3 – Arbre d'un schéma XML

Le langage XML a aidé à l'évolution de l'information sur le Web. En effet, l'information est devenu plus accessible et le stockage moins couteux.

1.4 Techniques utilisées

Le volume des sources de données double chaque année. La plupart de ces sources se trouvent au sein des réseaux comme les Intranet, extranet ou encore, le réseau des réseaux, Internet. De plus, près de 80% d'entre elles sont non structurées. A leurs échelle, ces documents sont la première source d'informations de leurs groupes d'utilisateurs. Face à cette densité en ligne, il est de plus en plus difficile d'exploiter efficacement ces informations. En effet, l'abondance de ces ressources, leur perpétuelle évolution et l'aspect polymorphe de leur contenu (format, média, type d'utilisation) sont autant d'écueils qu'il faut contourner. La nécessité de disposer de méthodes et d'outils avancés permettant d'y remédier [10].

Une panoplie de méthodes a vu le jour (Les robots d'indexation, Web sémantique, X-Path, Web Mining, ...). Dans ce qui suit nous expliquons les plus importantes parmi elles.

1.4.1 Robots d'indexation

Pour indexer de nouvelles ressources, un robot procède en suivant récursivement les hyperliens trouvés à partir d'une page pivot. Par la suite, il est avantageux de mémoriser l'URL de chaque ressource récupérée et d'adapter la fréquence des visites à la fréquence observée de mise à jour de la ressource.

Un fichier d'exclusion (robots.txt) placé dans la racine d'un site web permet de donner aux robots une liste de ressources à ignorer. Cette convention permet de réduire la charge du serveur web et d'éviter des ressources sans intérêt. Par contre, certains robots ne se préoccupent pas de ce fichier.

Deux caractéristiques du Web compliquent le travail du robot d'indexation : le grand volume de données et la bande passante. Un très grand nombre de pages sont ajoutées, modifiées et supprimées chaque jour. Si la capacité de stockage d'information, comme la vitesse des processeurs, a augmenté rapidement, la bande passante n'a pas bénéficié de la même progression. Le problème est donc de traiter un volume toujours croissant d'information avec un débit limité. Le robot a donc besoin de donner des priorités à ses téléchargements [22].

inconvénients

Malgré la grande utilisation de cette technique, néanmoins elle présente quelques inconvénients :

Ce n'est pas parce qu'une ressource a été servie par le serveur qu'elle a été reçue par le client. De plus, il est très difficile d'identifier avec précision un même utilisateur. En effet, l'adresse IP et les cookies généralement employés pour ce faire ne sont pas des garanties suffisantes quant à l'identité d'un visiteur. Par ailleurs, certains systèmes confondent certains robots automatisés avec des utilisateurs humains.

De plus, les webmasters ayant un plein pouvoir sur les données à l'origine des analyses, ils peuvent les manipuler. Cela peut se faire notamment en dupliquant les données de fréquentation, en ajoutant plusieurs marqueurs sur une même page ou en forçant un rafraîchissement intemsif des pages.

1.4.2 Web sémantique

Le Web sémantique désigne un ensemble de technologies visant à rendre le contenu des ressources du World Wide Web accessibles et utilisables par les programmes et agents logiciels.

Le Web sémantique, proposé initialement par le W3C [2], est d'abord une nouvelle infrastructure devant permettre à des agents logiciels d'aider plus efficacement différents types d'utilisateurs dans leur accès aux ressources sur le Web (sources d'information et services). Différents langages de niveau de complexité croissante sont proposés afin de mieux exploiter, combiner et raisonner sur les contenus de ces ressources. Les connaissances utilisées, par exemple sous forme de marqueurs sémantiques, doivent s'appuyer sur des ontologies afin de pouvoir être partagées et munies d'interprétations opérationnelles. La notion de méta-données est au coeur de la démarche avec une grande diversité dans l'interprétation et l'utilisation de cette notion. L'intégration automatique d'informations provenant de sources hétérogènes est cruciale particulièrement pour des applications d'entreprise [18].

inconvénients

Le web sémantique est une technique toujours en cours de recherche mais sa pratique presente quelques inconvénients :

- Tâche titanesque (sémantiser le web)
- Modélisation de la connaissance complexe
- Problème de mise à jour en continu de la connaissance
- ne comprend que les documents et données sémantiques

1.4.3 X-Path

Un langage d'interrogation des documents XML. Il permet de sélectionner certaines parties d'un document XML : des sous arbres, des nœuds, des attributs. Toutes applications ayant besoin de faire une recherche de fragment dans un document XML peut utiliser ce langage.
L'inconvénient est qu'il n'y a aucun moyen de filtrer ou d'interroger cette

2. World Wide Web Consortium, un organisme de standardisation fondé en octobre 1994, consortium chargé de promouvoir la compatibilité des technologies telles que HTML, XHTML, XML, RDF, CSS, PNG, SVG et SOAP

sortie avant le chargement du flux entier dans un document XML, même si uniquement un ou deux nœuds vous intéressent dans cette sortie.

1.4.4 Web Mining

Le Web mining étend la problématique du data mining aux données dépourvues de toute structure, aux données semi-structurées, ainsi qu'aux données évolutives. A un niveau plus général, il pose la problématique de la complexité opératoire et aussi systémique des outils à développer dans ce contexte. Cette complexité est très fortement liée à la masse importante des données du Web, leur diversité, leur distribution, leur aspect dynamique et au caractère ouvert du Web [10].

inconvénients

Le web mining travail essentiellement sur :

Le fichier log : cette technique utilise le fichier log pour extraire des données pertinente dont elle on aura besoin dans ses traitement, et cela pose un problème car ses fichiers sont très grands alors le calcul est important.

Les cookies : apportent beaucoup d'informations importantes aux serveurs mais il existe un refus ou suppression possible du cookie par l'internaute, un blocage possible par un pare-feu et finalement les cookies identifie un ordinateur et pas une personne [20].

1.5 Conclusion

Les techniques décrites précédemment sont très utilisées dans le Web, mais vu les inconvénients que présentent ces différentes techniques, nous nous proposons d'exploiter une nouvelle approche basée sur la théorie des arbres.

En se basant sur la théorie des arbres, on construit des automates d'arbres pour les documents XML ou les schémas XML ou les DTDs.

Le major avantage des automates d'arbres classiques qui représente un modèle très puissant (ayant un fondement mathématique très conséquent) est de pouvoir modéliser les documents XML.

Chapitre 2

Langages d'arbres rangés

Dans ce chapitre on rappel les automates sur les mots, puis on introduit les automates d'arbres ainsi que leurs techniques de déterminisme et de minimisation.

Les arbres sont des objets fondamentaux en informatique. Donald Knuth décrit les arbres comme " The most important non-linear structures that arise in computer algorithms " [11]. Un mot peut être vu comme un arbre particulier dans lequel chaque noeud a au plus un fils. Un tel arbre est dit unaire. Les arbres binaires sont des modèles pour lequel chaque noeud a 0 à 2 fils. Deux types de langages d'arbres sont distingués : les langages d'arbres de rang borné et les langages d'arbre de rang non-borné. Les noeuds des arbres d'un langage d'arbre de rang borné ont un nombre de fils borné par une constante qui dépend du langage alors que dans un langage d'arbres de rang non-borné, chaque noeud peut avoir un nombre arbitraire de fils. L'étude des automates sur les arbres de rang borné se reduit au cas des automates sur les arbres binaires [19].

Les automates d'arbres sont utilisés dans plusieurs champs de l'informatique fondamentale, en apprentissage, en réécriture, en démonstration automatique de théorèmes, en vérification de programmes, en bases de données et en pattern-matching. Ils peuvent également être à la base naturelle des modèles de traitement des documents XML [2]

2.1 Préliminaires

Dans cette partie on commence par définir les ensembles, mots et les langages ainsi que les automates sur les mots.

2.1.1 Ensembles, mots et langages

Si C est un ensemble fini, alors $|C|$ dénote le cardinal de C. Un alphabet est un ensemble fini de symboles. Un mot w (ou une chaîne de caractères) sur un alphabet Σ est composé de 0 ou plusieurs symboles de Σ. Le symbole ϵ dénote le mot vide qui est composé de 0 symbole. L'ensemble infini de tous les mots finis sur l'alphabet Σ est noté par Σ^*. Tout au long de notre travail nous utilisons le terme mot pour désigner une chaîne de caractères ou une séquence de symboles appartenant à un alphabet fixé Σ.

Soit w un mot (ou chaîne de caractères). Nous supposons que la première position de w est la position 0 et que $|w|$ dénote la taille (nombre de positions) de w.

- $w[i]$ est le i-ème caractère (symbole) de w
- $w[i:j]$ représente les caractères de la i-ème jusqu'à la j-ème position (inclus) de w
  $w[i:j] = w[i]w[i+1]\dots w[j])$, et
- $w[i:j] = \epsilon$, si $i > j$.

Si u et v sont des mots sur un alphabet Σ, alors leur concaténation uv (ou $u.v$) est encore un mot sur Σ. Un mot u est un préfixe du mot v s'il existe un mot w tel que $uw = v$. De la même manière, un mot u est un suffixe du mot v s'il existe un mot w tel que $wu = v$.

Un langage est un sous-ensemble de Σ^*. Les opérations suivantes sont définies sur les langages :

- La concaténation de deux langages L_1 et L_2 est définie comme le langage $L_1.L_2 = \{uv \mid u \in L_1 \wedge v \in L_2\}$.
- L'intersection de deux langages L1 et L2 est définie comme le langage $L_1 \cap L_2 = \{w \mid w \in L_1 \wedge w \in L_2\}$.
- L'union de deux langages L1 et L2 est définie comme le langage $L1 \cup L_2 = \{w \mid w \in L_1 \vee w \in L_2\}$.
- La fermeture L* d'un langage L est définie comme le langage.

$$L^* = \sum_{i=0}^{\infty} L^i$$

où L^0 contient seulement le mot vide ϵ et L^i les mots de taille $i > 0$, i.e., Pour tous les mots w. Tel que $w \in L^i, |w| = i$.
- La fermeture positive L^+ d'un langage L est définie comme le langage

$$L^+ = \sum_{i=1}^{n=\infty} L^i$$

2.1.2 Automate sur les mots

Dans ce qui suit le vocable automate désigne les automates à état finis.

Définition

Un automate à états fini est défini par le quintuplé $A = (Q, \Sigma, \Delta, I, F)$ Tel que :
- Q : un ensemble fini d'états
- Σ : un alphabet
- Δ : une fonction de transition T : $\Sigma \times Q \to Q$
- $I \subseteq Q$ l'ensemble des états initiaux,
- $F \subseteq Q$ l'ensemble des états terminaux.

2.1.3 Représentation

Un automate à états fini est représenté par un graphe orienté dont les nœuds représentent les états et les arcs, annotés par des éléments de l'alphabet, décrivent les transitions [17].

Exemple

$L = a^*b$

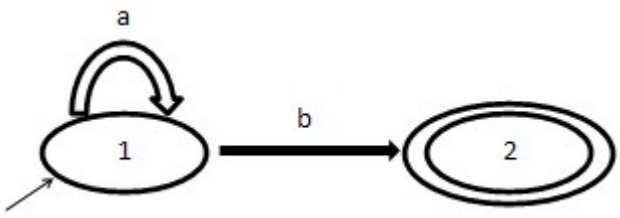

FIGURE 2.1 – Automate du langage a*b

2.1.4 Mot reconnu par un langage

La lecture d'un mot démarre du caractère avec ($pos = 0$) par l'intermédiaire de l'état initial, le mot est reconnu si tous les caractères ont été lu et que l'état auquel aura abouti est un état final.

$L(A)$ un langage reconnu par un automate A est l'ensemble de tous les mots (termes) acceptées par cet automate

2.1.5 Généralisation

Alphabet

L'alphabet pour un arbre est défini par le couple (Σ, r)
Tel que :

- Σ : ensemble de symbole (Une suite de symbole tout comme pour les automates des mots).
- r : arité de chaque symbole (le nombre de fils de chaque nœud).

Exemple

$\Sigma = \{(a, 1), (b, 0), (c, 3)\}$.

Dans cet exemple Σ est un alphabet rangé (arité fixe).

- Le terme symbole représente un nœud de l'arbre
- Un symbole $(a, 0)$ est une constante (feuille). noté aussi $(a, 0) \equiv a$.
- Un symbole $(a, 1)$ est unaire. $(a, 1) \equiv a()$
- Un symbole $(a, 2)$ est binaire $(a, 2) \equiv a(,)$

Terme

Un terme d'arbre est le plus petit ensemble défini par

- Chaque feuille (constante) est un terme $[\forall (a, 0) \in \Sigma \;\; a \in T]$
- Si $a(, \ldots,)[(a, n)] \in \Sigma$ et $(t_1, t_2, \ldots, t_n) \in T$ Alors
  $\mathrm{a}(t_1, t_2, \ldots, t_n) \in T$

Exemple

$\Sigma = \{a, b, c()\}$
$T = \{\{a\}, \{b\}, \{c(a)\}, \{c(b)\}\}$
Un terme $t \in T$ est défini comme un élément de $t(\Sigma)$ à travers Σ.

FIGURE 2.2 – Exemple de Terme

2.1.6 Arbres et domaines d'arbres

Domaine d'arbre

Un domaine d'arbres $D \subseteq N^*$ est défini par :
- D est fermé sur les préfixes si $u, u' \in N^*$ et $u' = u\delta$ noté $u \leq u'$ et $u' \in D$ alors $u \in D$
- Si $j \geq 0$ et $u_j \in D$ et $0 \leq i \leq j$ alors $u_i \in D$

Exemple

- D = $\{\epsilon$,0,1,(0,0),(0,1)$\}$ est un domaine d'arbre
- D = $\{\epsilon$,0,(0,0),(1,1)$\}$ n'est pas un domaine d'arbre car $((1,1) \in D$ mais $1 \notin D$

Arbre

Un arbre t est défini comme suit : $t = (D, l)$
tel que :
- D : position du nœud
- l : Etiquetage

Exemple

$T = \{(\epsilon, a), (0, b), (1, c), (2, d), (00, e), (01, f), (02, g),$
$(10, h), (20, j), (21, i), (010, k), (011, l), (012, m)\}$

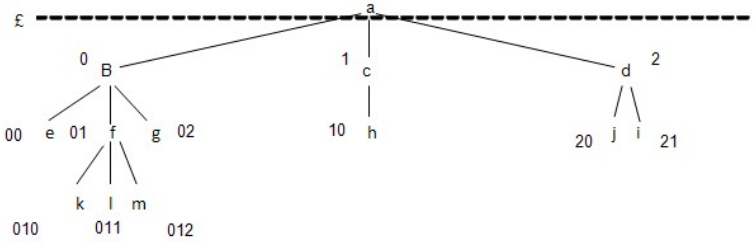

FIGURE 2.3 – Exemple d'un arbre

Domaine d'application

(K, L)Testable : Un des exemples de l'utilité des domaines d'arbre sont les langages (K, L)Testable, c'est une fenêtre de K voisins avec une profondeur L pour un nœud donné, cette fenêtre peut parcourir un arbre afin de le partitionner. Par exemple la vérification de l'état de similarité entre deux arbres. Ces langages peuvent facilement être interprétés via les domaines d'arbre [8].

2.2 Automates d'arbres

Un automate d'arbre est un graphe algébrique permettant de stocker, calculer et reconnaître plusieurs arbres. Il existe deux types d'automates d'arbres (descendant et ascendant).

Descendant

Un automate d'arbre descendant est représenté par le quadruplet $A = \{Q, \Sigma, I, \Delta\}$
Tel que :

- Q : l'ensemble des états
- Σ : l'alphabet $\{(\Sigma,$ r$)\}$
- I : ensemble des états initiaux $(I \subseteq Q)$
- Δ : ensemble de règles de transition de la forme : $q \to f(q_1, q_2, \ldots, q_n)$ avec $q, q_1, \ldots, q_n \in Q, f(, \ldots,) \in \Sigma$

Exemple

On reprend l'arbre de la Figure 2.3, l'automate sera comme suit :

- $Q = \{q_1, q_2, q_3, q_4\}$
- $\Sigma = \{a(,), b(,), c, d, e\}$
- $I = \{q_1\}$
- $D = \{q_1 \rightharpoonup a(q_1, q_2), q_1 \to b(q_3, q_4), q_2 \to c, q_3 \to d, q_4 \to e\}$

Limite des automates d'arbres descendant

Comme pour les automates classiques, les automates d'arbres finis peuvent être déterministes ou pas. Suivant la façon dont les automates se "déplacent" sur l'arbre qu'ils traitent, les automates d'arbres peuvent être de deux types

(ascendants/ descendants). La distinction est importante, car si les automates non-déterministes (ND) ascendants et descendants sont équivalents, les automates déterministes descendants sont strictement moins puissants que leurs homologues déterministes ascendants, car les propriétés d'arbres spécifiées par les automates déterministes descendants ne peuvent dépendre que des propriétés de chemins [21].

2.2.1 Ascendant

Définition

Un automate d'arbre d'états finis non-déterministe ascendant AAEFND est représenté par le quadruplet
$A = \{Q, \Sigma, Q_f, \Delta\}$
Tel que :

- Q : l'ensemble des états
- Σ : l'alphabet $\{(\Sigma, r)\}$
- Q_f :Ensemble des états finaux $Q_f \subseteq Q$
- Δ : Ensemble de règles de transition de la forme : $f(q_1, q_2, \ldots, q_n) \to q$ avec $q, q_1, \ldots, q_n \in Q, f(, \ldots,) \in \Sigma$

Exemple

On reprend l'arbre de la figure 2.3, son automate sera comme suit :

- $Q = \{q_1, q_2, q_3, q_4, q_f\}$
- $\Sigma = \{a(,), b(,), c, d, e\}$
- $Q_f = \{q_f\}$
- $\Delta = \{d \to q_1, e \to q_2, b(q_1, q_2) \to q_3, c \to q_4, a(q_3, q_4) \to q_f\}$

Calcul, reconnaissance d'un terme (arbre)

La fonction de calcul [1]d'un arbre t par un automate A est définie comme suit :

$$A(t) = \begin{cases} \Delta(f) \text{ si} f \in \Sigma_0 \\ \Delta(f(A(t_1), \ldots A(t_n))) si f(t_1, t_2, \ldots t_n) \in \Sigma_n \end{cases}$$

1. le vocable calcul ou execution sont utilisés pour le même sens dans ce mémoire

Exemple

On utilise l'arbre de l'exemple précèdent. $t = a(b(d, e), c)$
$A(d) = q_1$
$A(c) = q_4$
$A(e) = q_2$
$A(b(d, e)) = \Delta(b(A(d), A(e)) = \Delta(b(q_1, q_2) = q_3$

$A(a(b(d, e), c)) = \Delta(a(A(b(d, e)), A(c)) = \Delta(a(q3, q4)) = qf$

Un arbre est accepté si $A(t)$ existe et que $A(t) \in Q_f$

2.2.2 Langage d'arbre reconnaissable par un automate

Un langage d'arbre L est un sous-ensemble de $\mathrm{T}(\Sigma)$
- $\mathrm{L} \subseteq \mathrm{T}(\Sigma)$

Un langage d'arbre $L(A)$ reconnaissable par un automate A est l'ensemble de tous les termes acceptés par A. Un ensemble de termes L est reconnaissable par un automate d'arbres A si :
- $L = L(A)$

2.2.3 Déterminisme

Généralement les automates sont non-déterministes, en effet un déplacement unique dans les règles de transitions n'est pas assuré en plus aucune application ou prédicat ne guide le choix de la bonne règle transition à appliquer, le parcours de toutes les possibilités peut emmener à des chemins avec impasse, ce parcours est souvent NP-complet d'où l'obligation de transformer les automates non-déterministe en automates déterministe.

Automate d'arbre déterministe

Dans les automates sur les mots, on n'utilise pas les automates non-déterministes à cause du besoin d'avoir un seul chemin. Même chose pour les automates d'arbre, leurs application nécessite un résultat unique et s'est pour cela qu'on utilise les automates d'arbre déterministe.

Etat accessible

Un état q est dit accessible si et seulement s'il existe un terme t avec $A(t) = q$

Automate complet

On dit qu'un AAEFND est complet s'il existe au moins une règle $f(q_1, \ldots, q_n) \to q \in \Delta$ pour chaque $n \geq 0, f \in \Sigma$ et $q_1, \ldots, q_n \in Q$.

Automate réduit

On dit qu'un AAEFND est réduit si et seulement si tous ses états sont accessibles. On présente ici un algorithme de réduction basé sur le calcul d'états accessibles et donc supprimer les états inaccessibles [4].

Algorithme de réduction d'AAEFND

entrée : AAEF $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$
début
$Marqué = \phi$ ($Marqué$ est l'ensemble des états accessibles)

Répéter
Si $\exists q \in Q, f \in \Sigma, q_1, \ldots, q_n \in Marqué, f(q_1, \ldots, q_n) \to q \in \Delta$
alors
$Marqué$=$Marqué \cup \{q\}$
FinSi
Jusqu'a Aucun état ne peut être ajouté à $Marqué$
$Q_r = Marqué$
$Q_{r\ f} = Q_f \cap Marqué$
$\Delta_r = \{f(q_1, \ldots, q_n) \to q \in \Delta | q, q_1, \ldots, q_n \in Marqué\}$
Fin
Sortie : AAEF $\mathcal{A}_r = (Q_r, \Sigma, Q_{rf}, \Delta_r)$

Exemple

On prend comme exemple l'automate ayant les règles suivantes :
a $\rightarrow q_1$
b $\rightarrow q_2$
c$(q_1, q_2) \rightarrow q_3$
d$(q_1, q_4) \rightarrow q_5$
f$(q_1, q_5) \rightarrow q_f$
g$(q_2, q_3) \rightarrow q_{f2}$

Exécution

Marqué $= \phi$
Marqué $= \{q_1\}$
Marqué $= \{q_1, q_2\}$
Marqué $= \{q_1, q_2, q_3\}$
Marqué $= \{q_1, q_2, q_3, q_{f2}\}$
$A_r = (\{q_1, q_2, q_3, q_{f2}\}, \Sigma, \{q_{f2}\}, \{a \rightarrow q_1, b \rightarrow q_2, c(q_1, q_2) \rightarrow q_3, d(q_1, q_4) \rightarrow q_5, f(q_1, q_5) \rightarrow q_f, g(q_2, q_3) \rightarrow q_{f2}\})$

Existence de l'automate déterministe

On peut grâce au déterminisme supprimer l'ambiguïté présente dans un AAEFND, car dans ce dernier, on peut trouver plusieurs manières de calculer un terme , néanmoins, le fait de rendre l'automate déterministe facilite le calcul et le rend rapide.

Théorème

Soit L un ensemble reconnaissable de termes clôt, alors il existe un AAEFD A avec : $L = L(A)$.

Algorithme de determinisme d'AAEFND

entrée : AAEFNDR $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$
début
$Q_d = \Phi$, $\Delta_d = \Phi$

Répéter
Si $f \in \Sigma_n, s_1, \ldots, s_n \in Q_d, s = \{q \in Q | \exists q_1 \in s_1, \ldots, q_n \in s_n, f(q_1, \ldots, q_n) \to q \in \Delta\}$
alors
$Q_d = Q_d \cup \{s\}; \Delta_d = \Delta_d \cup \{f(s_1, \ldots, s_n) \to s\}$
FinSi
Jusqu'a Aucune règle ne peut être ajouté à Δ_d
$Q_{d\ f} = \{s \in Q_d | s \cap Q_f \neq \Phi\}$
Fin
Sortie : AAEFD $\mathcal{A}_d = (Q_d, \Sigma, Q_{d\ f}, \Delta_d)$

2.2.4 Minimisation

Congruence

Une relation de congruence $\equiv$ sur un ensemble de termes $T(\Sigma)$ est toute relation d'équivalence qui assure la propriété suivante :

Pour tout $f \in \Sigma_n$: $u_i \equiv v_i, 0 \leq i \leq n \Rightarrow f(u_1, \ldots, u_n) \equiv f(v_1, \ldots, v_n)$

Soit $T, u, v \in T(\Sigma)$, $T[u \to v]$ note l'arbre obtenu en substituant u par v dans T

Soit $s, t \in T(\Sigma)$, $s \blacktriangleright t$: s est un sous-arbre dans t

Pour un langage d'arbres L, on définit la relation de congruence $\equiv_L$ sur $T(\Sigma)$ par :

$$u \equiv_L v \text{ si : } \left\{\forall t \in T(\Sigma), s \blacktriangleright t, t[s \to u] \in L \Leftrightarrow t[s \to v] \in L\right\}$$

La relation $\equiv_L$ fragmente l'ensemble L en sous-ensembles disjoints appelés L-classes ; on dit que deux termes son $\equiv_L$ distincts s'ils ne sont pas $\equiv_L$

Théorème de Myhill-Nerode

Les expressions suivantes sont équivalentes :

(i) L est un langage d'arbres reconnaissable.

(ii) L est l'union d'un ensemble fini de classes d'équivalence d'une relation de congruence $\equiv$

(iii) La relation $\equiv_L$ est une relation de congruence possédant un ensemble fini de classes d'équivalence [12].

Ce théoréme très puissant, nous permet plusieurs déductions

- De (i) Un langage d'arbre reconnaissable $\Rightarrow \exists$ automate d'arbre qui reconnait L.
- De (ii) On deduit qu'ensemble fini de $\equiv$ on peut fusionner les états qui reconnaissent deux termes appartenons à la même classe.
- De (i) et (ii) On peut minimiser n'importe quel automate d'arbre.
- De (iii) L'automate minimale existe et comme les classes ne peuvent pas être fragmentées (par définition) $\Rightarrow$ l'automate minimale est unique.

Algorithme de Minimisation

Entrée : AAEFD $\mathcal{A} = (Q, \Sigma, Q_f, \Delta)$
début
$P = \{Q, Q - Q_f\}$

Répéter
$P' = P$
Pour Tout $(q, q') \in P, f \in \Sigma_n, q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_n \in Q$:
Si$\Big(\Delta(f(q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_n)) \sim_P$
$\Delta(f(q_1, \ldots, q_{i-1}, q', q_{i+1}, \ldots, q_n))\Big)$
alors
$q \sim_{P'} q'$
FinSi
Jusqu'a $P = P'$
$Q_{min} = Cq(P)$
$\Delta_{min} = \{f(Cq(q_1), \ldots, Cq(q_n)) \rightarrow Cq(\Delta(f(q_1, \ldots, q_n)))\}$
$Q_{min\ f} = \{Cq(q) | q \in Q_f\}$
Fin
Sortie : AAEFDM $\mathcal{A}_{min} = (Q_m in, \Sigma, Q_{min\ f}, \Delta_{min})$

2.3 Conclusion

Dans ce chapitre, un ensemble de définitions, généralités et algorithmes concernant les automates classiques a été rappelé. Les automates d'arbres, le vocabulaire et les notions qui leurs sont liées sont aussi exposés.

L'apport de ces modèles pour la modélisation des documents XML est traitée dans le chapitre suivant.

Chapitre 3

Langages d'arbres non-rangés

Nous avons vu que la modélisation des documents XML par un arbre est très naturelle et très puissante. Dans ce chapitre on mettra le point sur quel type d'arbre utilisé ainsi quel automate utiliser pour la reconnaissance [5] [2].

3.1 Problématique

Afin d'expliquer les problèmes qui sont liés à la modélisation des documents XML, nous allons utiliser des exemples illustratifs.

Exemple 1

```
<universite Nom="Université de Moscu">
 <Laboratoire>
   <Nom> Informatique </Nom>
   <Chercheur CId="C001">
      <Nom> Victor M. Glushkov </Nom>
      <Titre> Professeur </Titre>
   </Chercheur>
   <Publication>
      <Sujet> Automates </Sujet>
      <Annee> 1961 </Annee>
      <Revue>
         <Nom> Revue Russe de Recherche </Nom>
         <TArticle idAuts="C001">
            THéorie Abstraite des automates
         <TArticle>
      </Revue>
```

</Publication>
</Laboratoire>
</Universite>.
Le document XML précédent est représenté par l'arbre suivant :

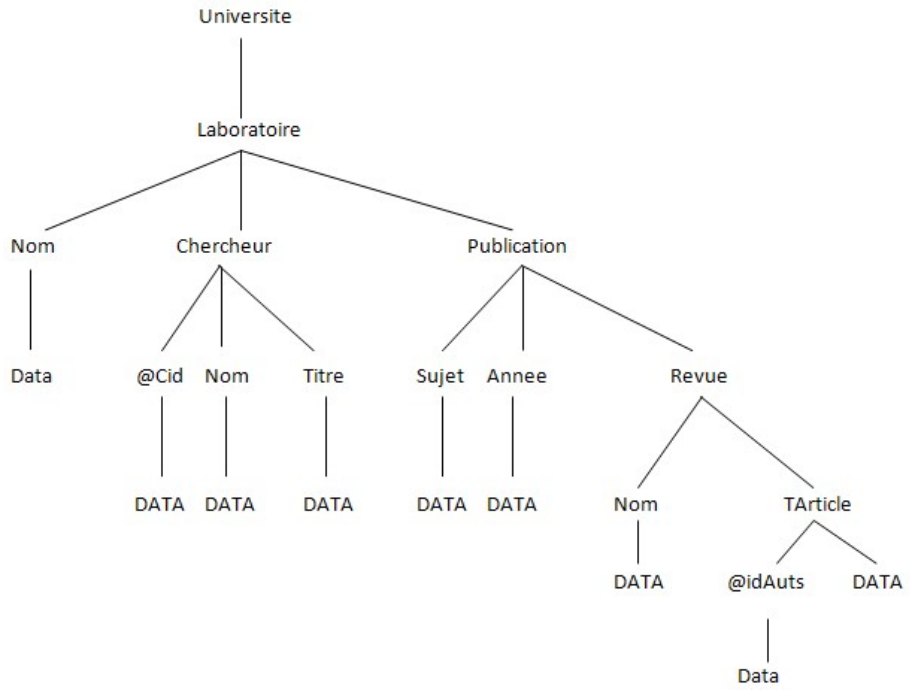

FIGURE 3.1 – Arbre de l'exemple 1

Ce type d'arbre peut être facilement représenté par un automate d'arbre classique. Σ = (universite, 1), (Nom, 1), (Chercheur, 3), (Publication, 3), (data, 0), (@Cid,1), (Titre, 1), (Sujet, 1), (Annee, 1), (Revue, 2), (TArticle, 2), (@idAuts, 1) Σ est un alphabet à arité fixe, il est dit un alphabet rangé.

Exemple 2

Soit la DTD suivante :

<!ELEMENT a (b*)>
<!ELEMENT b (a|b|c|d|*)>
<!ELEMENT c (#PCDATA)>
<!ELEMENT d (a ?b ?)>

L'arbre de cette DTD sera représenté comme suit :

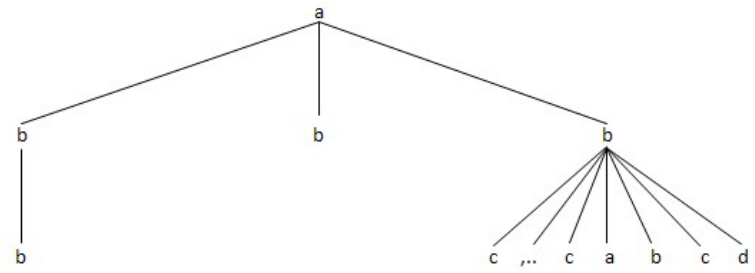

FIGURE 3.2 – Arbre de l'exemple 2

Constat 1

Dans cet exemple L'élément " b " peut être feuille, ou un terme unaire, ou bien un terme n-aire. Alors il est impossible de définir un alphabet rangé pour ce type d'arbres, et c'est le cas général des documents XML.

Pour attirer l'attention du lecteur sur un autre problème, nous avons besoin de donner certaine définitions concernant les arbres non-rangés. Ces concepts seront détaillés dans la prochaine section.

Alphabet

Dans la définition qu'on a utilisé précédemment le r représente l'arité de chaque nœud, si l'on veut utiliser les automates non-rangés nous nous pouvons plus fixé comme arité r, et chaque nœud est donc représenté seulement par un symbole Σ, Alors $a, a(), a(,), \ldots, a(,,\ldots,)$ sont toutes des expressions valides.

Terme

Un terme d'arbre est le plus petit ensemble défini par
- Chaque feuille (constante) est un terme [$\forall$(a, 0)$\in \Sigma$, a$\in$T]
- Si a (,...,), $a \in \Sigma$ et $(t_1, t_2, \ldots, t_n)\in$ T Alors
  a$(t_1, t_2, \ldots, t_n) \in T$
  Avec $n \geq 0$

Remarquons que la variable n est le seul changement apporté à la définition de l'alphabet d'un automate classique.

Après ces deux définitions, on peut donc généraliser toutes les définitions, propriétés et algorithmes des automates d'arbre rangés (à arité fixe) pour les documents XML.

Maintenant reprenant l'exemple 2 de la DTD.

< ! ELEMENT a (b*)>

b(expression)→ q_b est l'expression pour représenter $a(b^*)$

$a \to q_a$
$a(q_b) \to q_a$
$a(q_b, q_b) \to q_a$
$a(q_b, q_b, q_b) \to q_a$
...
$a(q_b, \ldots, q_b) \to q_a$
...

Constat 2

Le nombre infini de transitions limite l'utilisation des arbres classiques comme modèle pour les documents XML.

Après avoir mit le point, avec ces deux exemples sur les problèmes liés à la modélisation des documents XML. La question qu'on se pose est : quel type d'automate peut représenter ce type de document si les automates classiques ne peuvent pas le faire !

Pour cela on se propose d'utiliser les arbres non-rangés reconnu par les automates à haie encoder pour des soucis de simplification par des automates à pas.

En effet, dans ce qui suit nous rappelons quelques définitions nécessaires pour comprendre ces modèles.

3.2 Automate à Haie

3.2.1 Introduction

les automates à haie qui ont été introduits par Brüggemann-Klein, Murata et Wood [1] en 2001. Ils utilisent des arbres à arités arbitraires, et des séquences d'arbres appelées haies [9].

Etant donné un langage L, pour connaître les mots qu'il contient, il faut décrire L. Les grammaires formelles sont utilisées pour décrire les langages. La notion de grammaire formelle a été introduite par Chomsky. Une grammaire G est un quadruplet (N, Σ, P, I) où :

- Σ est l'alphabet des symboles terminaux (symboles apparaissant dans les mots générés)
- N est l'alphabet des symboles non-terminaux (symboles n'apparaissant pas dans les mots générés)
- $I \in N$ est le symbole initial
- P est un ensemble fini de règles de production de la forme $\alpha \rightarrow \beta$, où $\alpha \in N_\Sigma^+$ et $\beta \in N_\Sigma^*$ avec $N_\Sigma = (N \cup \Sigma)$

La signification intuitive de ces règles de production est que la suite non vide de symboles terminaux ou non-terminaux $\alpha \in N_\Sigma^+$ peut être remplacée par la suite, éventuellement vide, de symboles terminaux ou non-terminaux $\beta \in N_\Sigma^*$[5].

Une grammaire génère les mots d'un langage en réécrivant le symbole initial I. Si $\alpha \rightarrow \beta$ est une règle de production de la grammaire G, on peut réécrire (ou dériver) le mot $(a\alpha b)$ en $(a\beta b)$ en appliquant $\alpha \rightarrow \beta$ sur $(a\alpha b)$.

Chomsky a défini une classification des grammaires formelles en quatre types dépendants de la forme de leurs règles de production. Soient $\alpha\beta \in N_\Sigma^*, A \in N, v \in N_\Sigma^+$ et $a, b \in \Sigma$.

Type	Forme de la règle de production	Remarques
type0	α → β	Aucune restriction sur les règles de production
type1	αA → αvβ	Dépendante du contexte
type2	A → α	Indépendante du contexte (hors contexte)
type3	A → aB ou A → Ba	Régulière (équivalente aux expressions régulières)

FIGURE 3.3 – Classification de Chomsky

.

Les langages réguliers (type 3) sont équivalents aux expressions régulières [5].

3.2.2 Langages réguliers, Expressions régulières

Un langage régulier ou langage rationnel ou encore langage de type 3 dans la hiérarchie de Chomsky, est un langage formel que l'on peut définir grâce à une expression régulière sur un alphabet.

Expression régulière

Une expression régulière E sur un alphabet Σ et son langage associé $L(E)$ sont définis de façon inductive comme suit :

- 1. Les constantes ε et $\emptyset$ sont des expressions régulières et elles dénotent les langages $\{\varepsilon\}$ et $\emptyset$, respectivement, c'est-à-dire, $L(\varepsilon) = \{\varepsilon\}$ et $L(\emptyset) = \emptyset$.
- 2. Si $a \in \Sigma$, alors a est une expression régulière qui dénote le langage L(a)={a}.
- 3. Si E et F sont des expressions régulières, alors $E + F$ est une expression régulière (notée aussi par $E|F$) qui dénote l'union de $L(E)$ et $L(F)$, i.e., $L(E + F) = L(E) + L(F)$.
- 4. Si E et F sont des expressions régulières, alors EF est une expression régulière qui dénote la concaténation de $L(E)$ et $L(F)$, i.e., $L(EF) = L(E)L(F)$.
- 5. Si E est une expression régulière, alors E^* est une expression régulière qui dénote la fermeture de $L(E)$ (étoile de Kleene), i.e., $L(E^*) = (L(E))^*$.
- 6. Si E est une expression régulière, alors E^+ est une expression reégulière qui dénote la fermeture positive de $L(E), i.e., L(E^+) = L(E)L(E^*)$.
- 7. Si E est une expression régulière, alors $E?$ est une expression régulière qui dénote le langage L(E) qui de plus, accepte le mot vide, i.e., $L(E?) = L(E) \cup \{\epsilon.\}$
- 8. Si E est une expression régulière, alors (E), l'expression régulière avec parenthèses, est aussi une expression régulière qui dénote le même langage que E.

Pour éviter le recours excessif aux parenthèses, on pose que l'étoile de Kleene (et la fermeture positive +) a la plus haute priorité, suivie par la concaténation puis par l'union. Par exemple, $(ab)c$ peut s'écrire abc et $a|(b(c^*))$ peut s'écrire $a|bc^*$. Si on veut changer la priorité des opérateurs, alors on utilise les parenthèses.

Un langage L est régulier s'il existe une expression régulière E telle que $L = L(E)$. Un mot w est une instance d'une expression régulière si w appartient au langage défini par E. Le problème est de savoir comment déterminer si un mot w est dans un langage régulier L a été traité par Kleene : il a montré que les automates d'états finis et les expressions régulières définissent la même classe de langages [5].

3.2.3 Alphabet non-rangés

Dans le chapitre précédent, on a vu qu'un arbre rangé t est une application du domaine $D(t) \subseteq N^*$ vers un ensemble d'étiquettes L.
Pour définir la notion de non-rangement, on définit

$$\Sigma^+ = U \ \cup \ \bigcup_{i=0}^{n} \Sigma_i$$

Avec Σ un alphabet rangé, U ensemble de symbole representant un alphabet non-rangé

Extension du domaine d'arbre

Le domaine d'arbre sera défini comme suit :

- D est fermé sur les préfixes si $u, u' \in N^*$ et $u' = u\delta$ noté $u \leq u'$ et $u' \in D$ alors $ui \in D$
- $\forall p \in D(t)$ si $t(p) \in \Sigma_n$ alors
  Si $j \geq 0$ et $uj \in D$ et $0 \leq i \leq j$ alors $ui \in D$
  si $t(p) \in U \ \{j | P_j \in D(t) = \{1, \ldots, k\}$ avec k variable≥ 0

3.2.4 Une haie

Une haie a(h) est un arbre non-rangé
Tel que :

- a : est la racine de l'arbre.
- h : une concaténation de haies (arbre non-rangé) [16].

Exemple

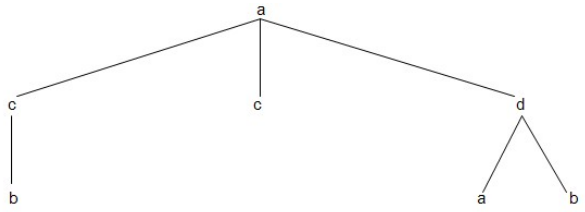

FIGURE 3.4 – Exemple d'une Haie

.

Représentation en haie : a(c(b) c d(a b)
Représentation en classique : (c(b), c, d(a b))

On remarque qu'une haie suit une forme d'une expression régulière, ce qui nous permet d'utiliser toute la puissance des expressions régulières dans les automates à haies.

Définition

Un automate d'arbre à haie A est défini par le quadruplé : $A = \{Q, \Sigma, Q_f, \Delta\}$ Tel que :

- Q : Ensemble fini d'état
- Σ : Un alphabet
- Q_f : Ensemble des états finaux $Q_f \subseteq Q$
- Δ : $a(R) \to q, a \in \Sigma, q \in Q$
  tel que :
  - R : Une expression régulière de Q [4]. [6].

Exemple

Soit $A = (Q, \Sigma, Q_f, \Delta)$ un automate à haie avec $Q = \{q, q_b, q_c\}, Q_f = \{q_c\}$, et Δ est définit par les règles suivantes :

- $a(Q^*) \to q$, $a(Q^* q_b Q^*) \to q_b$, $a(Q^* q_c Q^*) \to q_c$
- $b(Q^*) \to q_b$, $c(Q^* q_b Q^*) \to q_b$, $b(Q^* q_c Q^*) \to q_c$
- $c(Q^*) \to q_b$, $c(Q^* q_b Q^* q_b Q^*) \to q_c$, $c(Q^* q_c Q^*) \to q_c$

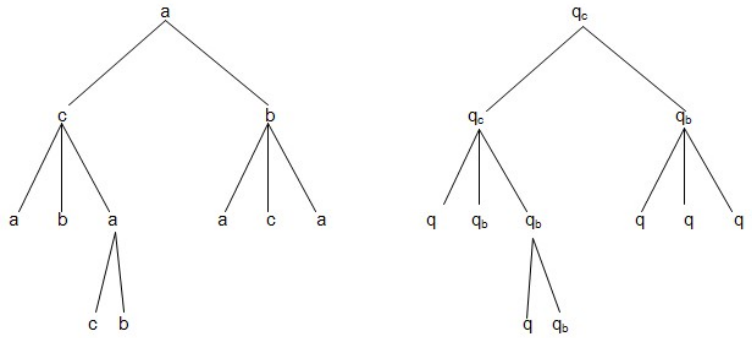

FIGURE 3.5 – Automate de A

Le côté droit de la figure montre une série de A. Considérons, par exemple, le premier fils de q_c. Il est marqué c dans l'arbre d'entrée. Dans la suite, la séquence de l'étiquette de son successeur est le mot qq_bq_b, Cela signifie que nous pouvons appliquer la règle $c(Q^* q_b Q^* q_b Q^*) \to q_c$ et donc obtenir q_c au nœud 1 par la suite [4].

Notons que la schématisation des automates à haie n'est pas possible à cause de l'infinité de "*".

3.2.5 Déterminisme

Dans le cas d'automates rangés un automate déterministe est un automate qui ne contient pas deux règles de transition avec la même partie gauche, cela implique que pour tout terme (arbre) il existe un seul chemin de calcul aboutissant à

$A(t) = q$ $(avec \in Q_f)$

Pour transférer cette propriété aux automates non-rangés. On doit assurer que pour tout état reconnaissant un arbre p qui doit être atteint via p ou un des successeurs de p.

Définition

Un automate à haie déterministe $A = (Q, \Sigma, Q_f, \Delta)$ est un automate à haie assurant la propriété suivante : $\forall \delta \in \Delta /$ Si $a(R_1) \rightarrow q_1$ et $a(R_2) \rightarrow q_2$ on a :

- $R_1 \cap R_2 = \emptyset$ ou bien $q_1 = q_2$

Exemple

L'automate de la Figure 3.5 n'est pas déterministe car il contient les deux règles de transitions :
- $a(Q^*) \rightarrow q$
- $a(Q^* q_b Q^*) \rightarrow q_b$

On remarque que l'union des deux parties gauches n'est pas vide. $(Q^*) \cap (Q^* q_b Q^*) = Q^* \neq \phi$[4].

Théorème

Pour tout automate à haie non-fini A, il existe un automate à haie fini et déterministe A_d acceptant le même langage. Le nombre d'états de A_d est exponentiel par rapport au nombre d'état de A [4].

Corollaire

La déterminisation d'un automate à haie est un problème NP-Complet.

3.2.6 Minimisation

Le problème de minimisation des automates d'arbre déterministes non-rangés est défini de manière analogue au problème de minimisation des automates finis : Etant donné un automate d'arbre déterministe non-rangés A et un entier m, décidant s'il existe un automate d'arbre déterministe non-rangés B tel que $L(B) = L(A)$ et la taille de B est inférieur ou égal à m. Les deux théorèmes affirme [14].
Wim Martens et Joachim Niehren demontre les deux théorémes suivant :

Théorème

l'automate minimal reconnaissant le même langage qu'un automate à haie n'est pas unique.

Théorème

l'obtention d'un automate minimum local pour un automate à haie est NP-complet.

Ces deux théorèmes affirment que non seulement la minimisation est une opération difficile en plus l'automate minimale n'est pas unique.

3.3 Automate à pas

A cause des problèmes de déterminisme et de minimisation l'implémentation des automates à haie reste toujours un problème, les chercheurs ont opté pour un nouvel encodage des automates : les automates à pas.

Pour définir les automates à pas on introduit la notion de langage horizontal pour une haie a(R), c'est l'ensemble des états associés à R.

L'idée fédératrice des automates à pas est d'introduire de nombreux états afin de calculer les langages horizontaux d'un automate à haie.

l'automate global de plusieurs automates classiques par niveau (pas), est dit automate à pas.

3.3.1 Définition

Un automate à pas pour un automate à haie est représenté par le quintuple $A_p = \{Q, \Sigma, \delta_0, Q_f, \Delta\}$
tel que :

- Q : Ensemble fini d'état
- Σ : Un alphabet
- $\delta_0 : \Sigma \to Q$ Est une fonction associant à chaque symbole de Σ un état initial.
- Q_f : Ensemble des états finaux $Q_f \subseteq Q$
- $\Delta : Q * Q \to Q$ est la fonction de transition [7].

Calcul

Pour chaque $a \in \Sigma$:

- $\Delta_a(wq) = \Delta(\Delta_a(w), q)$
- $\Delta_a(\varepsilon) = \delta_0(a)$

3.3.2 Exemple

Soit l'automate A ayant les règles de transition représentées sur Figure 3.6

La construction de l'automate à pas se fera par niveau en commençant des feuilles jusqu'à la racine.

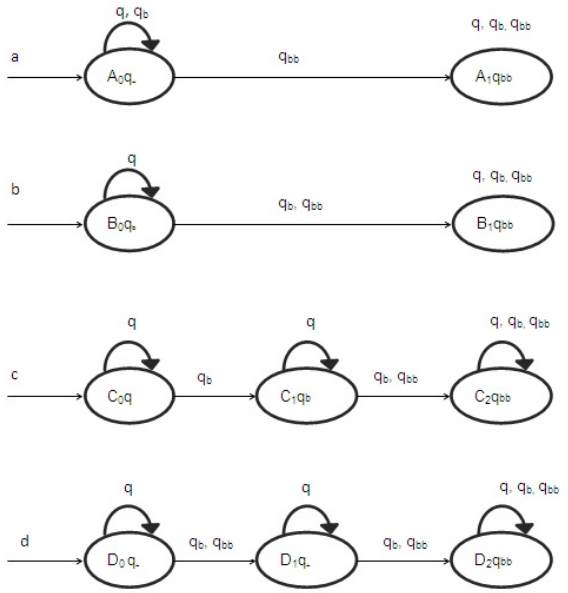

FIGURE 3.6 – Régles de transition de l'automate A

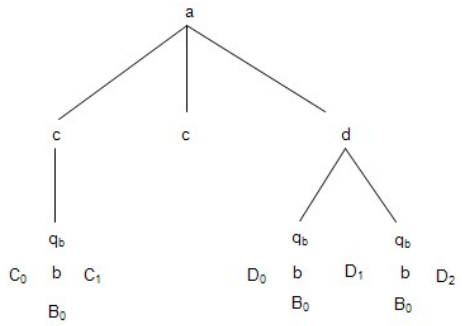

FIGURE 3.7 – Etape 1 du calcul de l'arbre

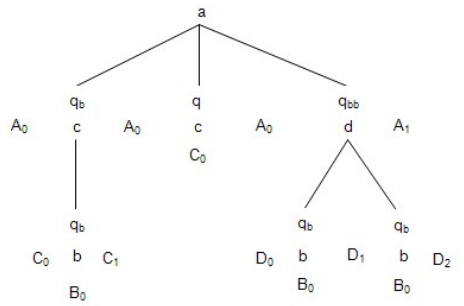

FIGURE 3.8 – Etape 2 du calcul de l'arbre

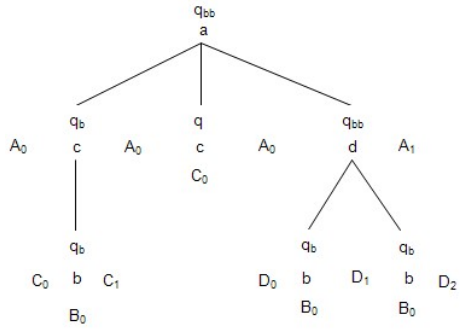

FIGURE 3.9 – Etape 3 du calcul de l'arbre

L'état de la racine (Symbole a) est $q_{bb} \in Q_f$, cet arbre est reconnu par l'automate à pas.

3.3.3 Déterminisme

Par définition 3.4.2 dans l'automate à pas, chaque symbole de l'alphabet Σ est associé à un état initial, ce qui veut dire que les automates à pas sont déterministes.

3.3.4 Minimisation

Pour chaque langage reconnaissable $L \subseteq T(\Sigma)$ il existe un unique automate à haie et à pas déterministe acceptant le langage L [4].

3.4 Conclusion

Dans le web le temps est une métrique très importante dans la recherche de donnée, malgré le problème de détérminisation les automates classique reste une méthode très puissante.

En pratique plusieurs simplifications sont utilisées pour rendre l'implémentation des automates d'arbres faisable. D'une part, le problème de l'arité variable est négligée par le rajout des règles de transitions, et d'une part les concepteurs fixe toujours un maximum pour implémenté l'operateur "*" ainsi réglé le problème de l'infinité des règles de transitions.

Un autre problème technique entrave ce puissant modèle c'est celui de la lourdeur de la détérminisation par des machines séquentielles, cette complexité peut toujours être allégé par l'usage de parallélisme des machines.

La minimisation est aussi coûteuse, cependant on peut la contourner et utilisé l'automate non minimal, ceci consommera plus de temps au calcul mais reste toujours acceptable.

Chapitre 4

Application

Dans la perspective de fournir un outil logiciel complet pour le traitement des documents XML pour l'usage des automates d'arbre non-rangés nous avons commencé par la conception et le développement d'une version préliminaire implémentant ainsi les opérations préliminaires et essentielles traitant les automates classiques.

Le but essentiel de cette partie expérimental est de se familiariser et prendre en main le développement d'un tel outil et de voir ce qui existe pour les comprendre voir les instrumenter.

Avant de décrire l'architecture de notre outil nous commençons par décrire des outils de base déjà existant et qui on fait leurs preuves dans ce domaine. Il s'agit de deux API SAX et DOM permettant de lire un document XML dans un fichier et d'un environnement de développement Visual Basic Express.

4.1 SAX

SAX est l'acronyme de Simple API for XML.

SAX est une API permettant de lire un fichier XML sous forme de flux. Le principe de fonctionnent est le suivant. L'application crée un parseur et elle enregistre auprès de ce parseur son gestionnaire d'événements. Au cours de la lecture du fichier contenant le document XML, le gestionnaire reçoit les événements générés par la parseur. Le document XML n'est pas chargé en mémoire [3].

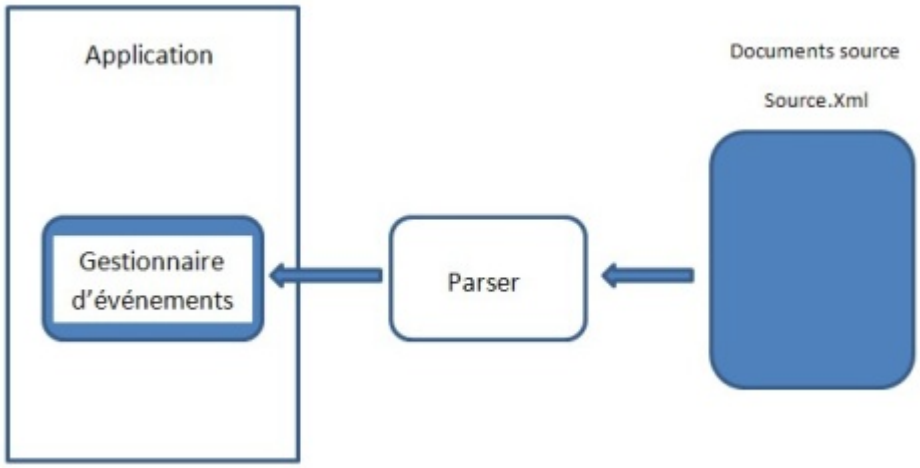

FIGURE 4.1 – Architecture de SAX

4.2 DOM

DOM est l'acronyme de Document Object Model.

DOM est une API permettant de charger un document XML sous forme d'un arbre qu'il est ensuite possible de manipuler. Le principe de fonctionnent est le suivant. L'application crée un constructeur qui lit le document XML et construit une représentation du document XML sous forme d'un arbre [3].

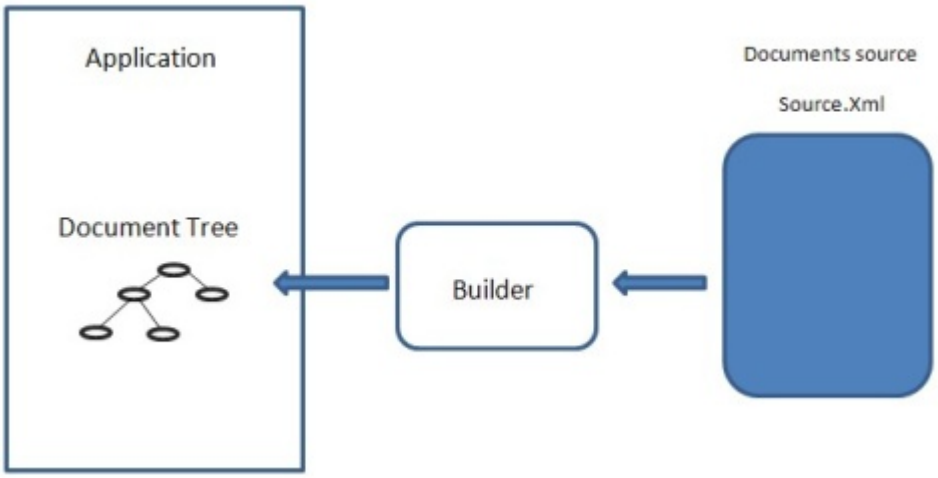

FIGURE 4.2 – Architecture de DOM

4.2.1 Comparaison de SAX et DOM

La grande différence entre les API SAX et DOM est que la première ne charge pas le document en mémoire alors que la seconde construit en mémoire une représentation arborescente du document. La première est donc particulièrement adaptée aux (très) gros documents. Par contre, elle offre des facilités de traitement plus réduites. Le fonctionnement par événements rend difficiles des traitements non linéaires du document. Au contraire, l'API DOM rend plus faciles des parcours de l'arbre.

4.3 Visual Basic Express

Microsoft Visual Studio Express est un ensemble d'environnements de développement intégrés gratuits développé par Microsoft. Il s'agit d'une version allégée de Microsoft Visual Studio. selon Microsoft, L'idée de ces éditions "express" est de fournir un environnement de développement facile à utiliser et à apprendre pour des jeunes ou des passionnés de developpement.

Dans notre phase de développement nous avons choisi de travailler avec Visual Basic Express, grâce à sa licence gratuite ainsi que sa rapidité pour le prototypage et le développement, et notre besoin été de trouver un langage capable de faire des manipulations rapide et facile de documents XML.

4.3.1 XmlTextReader

Le XmlTextReader se base sur les concepts précédemment définies SAX et DOM.
La classe XmlTextReader est une implémentation de XmlReader qui fournit un analyseur rapide et performant. Elle applique les règles selon lesquelles le code XML doit être correctement construit. Cet analyseur ne peut être consideré comme outil de validation puisqu'il ne possède pas d'informations DTD ou de schéma. Il peut lire du texte dans des blocs ou des caractères à partir d'un flux [15].

Fonctions

Parmis les fonctions du XmlTextReader lequels on a utilisé.

- AttributeCount :Obtient le nombre d'attributs du nœud actuel.(Substitue XmlReader.AttributeCount)
- Depth :Obtient la profondeur du nœud actuel dans le document XML.(Substitue XmlReader.Depth.)
- HasAttributes :Obtient une valeur indiquant si le nœud actuel possède des attributs.(Hérité de XmlReader.)

- LocalName :Obtient le nom local du nœud actuel.(Substitue XmlReader.LocalName.)
- Value Obtient le texte du nœud actuel.(Substitue XmlReader.Value.)
- GetAttribute :Si l'index est entier renvoi la valeur de l'attribut actuel.
  Si l'index est un texte renvoi la valeur de l'attribut avec un nom specifique.
- IsStartElement :indique si le nœud actuel est une balise de debut ou bien un element vide.
- Read :Lit le nœud suivant du flux.(Substitue XmlReader.Read.)

4.4 Déscription de l'outil developpé

XML Automata tool contient Les opérations essentielles suivantes :

- Parser un Doc XML : parser (Algorithme qui permet d'analyser un texte et d'en déterminer sa structure syntaxique afin d'effectuer divers traitement, comme par exemple une compilation.) le XmlTextReader est le responsable de cette opération.
- Définir un automate pour un Document XML : cette opération a comme entrée le document XML, elle définit l'automate correspondant en donnant tous les ensembles du Quadruplé représentant un automate classique.
- Définir un automate pour plusieurs Document XML : cette opération aura comme entrée un ensemble de document XML, et en sortie on aura l'automate correspondant à cet ensemble.
- Module de visualisation de l'arbre des documents XML, ainsi que les documents eux mêmes.

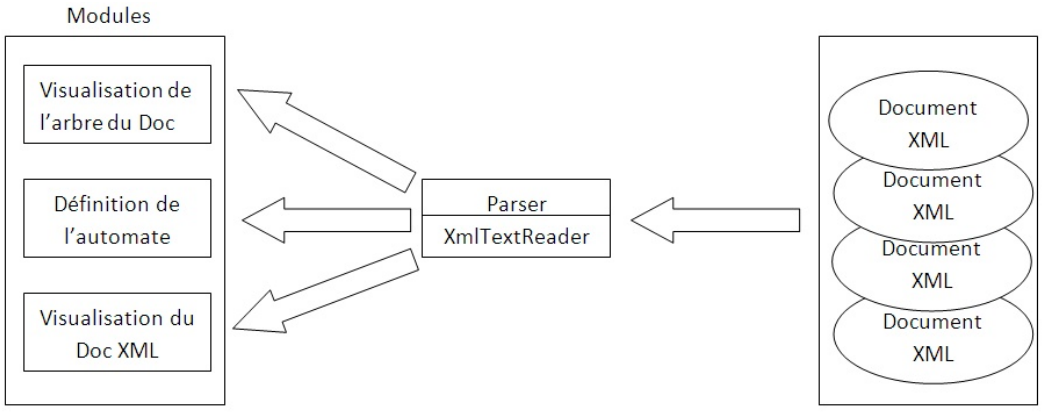

FIGURE 4.3 – Architecture de l'outil

4.5 Conclusion

La programmation des automates d'arbre demande beaucoup de patiente et une bonne compréhension des méthodes et des constructeurs, et l'utilisation des automates d'arbre pour la modélisation des documents XML demande une très bonne compréhension du langage XML ainsi que toutes opérations sur ses documents.

Prochainement on va doter notre outil de plus de fonctionnalité, pour qu'il supporte plusieurs type d'automate d'arbre et enfin être utilisé à la comparaison des performance de différentes méthodes.

Conclusion

Notre travail nous a permis de découvrir un grand axe qui est l'informatique fondamental, dans cet axe il existe plusieurs domaines de pratique et de recherche.

Les automates d'arbres sont un outil très puissant qu'on peut appliquer sur n'importe quelle masse de donnée et notamment le Web, en effet, un grand pourcentage des documents sont structurés en XML.

Les méthodes basées sur un fondement mathématique tel que les automates d'arbres ont plus de certitude et de fiabilité dans leurs résultats car leurs méthodes sont prouvées mathématique et ne pas par une simulation qui ne peut engendrer tout les cas de figure de l'application.

L'informatique fondamental m'a permis d'avoir une vision plus mathématique, et m'a poussé aux preuves mathématique pour ne laisser aucun cas au hasard.

Ce modeste travail est une initiation aux instruments de la théorie des arbres pour la modélisation des documents XML. Dans nos travaux futurs, nous envisageons d'explorer et étudier d'autres types d'automates.

D'autres travaux sont aussi envisagés :
- Un autre axe à explorer est celui de l'amélioration de la complexité des opérations de déterminisation et de minimisation.
- L'outil développé est aussi basique et est a enrichir par d'autres fonctionnalités.
- Mesurer l'efficacité de l'outil par rapport à l'existant.

Bibliographie

[1] M.Murata A. Brüggemann-Klein and D.Wood. Regular tree and regular hedge languages over unranked alphabets. *The Hongkong University of Science and Technology*, 2001.

[2] Béatrice Bouchou. *Validation et automates d'arbres.* Université François Rabelais Tours, Fac de Sciences, 2008.

[3] Olivier Carton. *L'essentiel de XML.* Université Paris Diderot, 2007.

[4] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on : `http://www.grappa.univ-lille3.fr/tata`, 2008. release October, 12th 2008.

[5] DENIO DUARTE. *Une méthode pour l'évolution de schémas XML préservant la validité des documents.* Thèse de doctorat, UNIVERSITE FRANCOIS-RABELAIS - TOURS, 2005.

[6] Florent Jacquemard et AL. On hedge automata languages and hedge rewriting. *COPS Meeting*, 2002.

[7] Julien Carme et AL. Querying unranked trees with stepwise tree automata. *International conf. on rewriting techniques and applications*, 2004.

[8] Pedro Garcia et AL. Learning k-testable and k-piecewise testable languages from positive data. *Université de Valence. Espagne*, 2004.

[9] Emmanuel Filiot. Automates d'arbres à arités arbitraires. Technical report, Ecole Normale Supérieure de Lyon, 2003.

[10] Jérémy Clech Salima Hassas. Web mining. http ://www710.univ-lyon1.fr/˜ hassas/ClechHassas.htm, consulté Avril 2011.

[11] Donald E. Knuth. *Fundamental Algorithms, volume 1 of The Art of Computer Programming.* Addison-Wesley, third edition, 1997.

[12] Dexter Kozen. On the myhill-nerode theorem for trees. *bull. Europ. Assoc. Theor. Comput. Sci. 47(June 1992), 170-173*, 1992.

[13] Comprendre l'ordinateur. C'est quoi XML ? http ://sebsauvage.net/comprendre/xml, consulté avril 2011.

[14] Wim Martens and Joachim Niehren. Minimizing tree automata for unranked trees. *Springer-Verlag Berlin Heidelberg*, 2005.

[15] Microsoft. Lecture de données XML avec XmlTextReader. http ://msdn.microsoft.com/fr-fr/library/aa720466%28v=vs.71%29.aspx, consulté Juin 2011.

[16] Makoto Murata. Hedge automata : a formal model for XML schemata, rapport tachnique. *Fuji Xerox information Systems*, 1999.

[17] Alexandre Pauchet. *Modélisation des Systèmes Complexes Introduction aux Automates à états finis*. INSA Rouen - Département ASI, 2009.

[18] Jean Charlet Philippe Laublet, Chantal Reynaud. Sur quelques aspects du web sémantique. *Université de Paris-Sorbonne*, 2002.

[19] Mathias Samuleides. Automates d'arbres à jeton. Thèse de doctorat, Université Paris 7, 2007.

[20] Stéphane Tufféry. *DATA MINING ET STATISTIQUE DÉCISIONNELLE*. 2006.

[21] Wikipedia. Automate d'arbres. http ://fr.wikipedia.org/wiki/Automate_d'arbres, consulté Avril 2011.

[22] www.techno science.net. Robot d'indexation. http ://www.techno-science.net/ ?onglet=glossaire&definition=356, consulté Avril 2011.