République Algérienne Démocratique Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Amar Telidji – Laghouat

Faculté des sciences

Département de mathématique et informatique

Thème

Algorithmique de recherche de motif dans les documents XML

Pour l'obtention du diplôme de Master en informatique

Spécialité : Système d'information et décision

Dirigé par : Guellouma Younes

Rédigé par : Brihmat Ilyes

Année universitaire 2013-2014

Remerciements

En premier lieu je tiens à remercier Dieu, le tout puissant, qui m'a donné le courage et la volonté pour assumer et terminer ce travail.

En témoignage de reconnaissance JE tiens à exprimer ma grande reconnaissance et mes profonds remerciements à mes enseignants en informatique et surtout mon professeur Madame kerrouche Badra pour m'avoir apporté une formation bénéfique et une rigueur scientifique et qui m'a tant aidé à trouver l'encadreur.

Je tiens à remercier mon encadreur Monsieur Guellouma Younes pour son soutien, sa confiance et ses précieux conseils qui m'ont facilité le chemin de cette étude.

Je tiens aussi à remercier Les membres du jury pour avoir accepté et apprécié ce travail.

A tous ceux qui, par leur aide ou leur soutien, ont contribué à la réalisation du présent travail de mémoire.

Dédicace

Je dédie le présent travail, à mes parents à mes frères et sœurs et à tous les amis.

Résumé

Le but de ce travail est l'étude d'un certain nombre d'algorithmes de recherche de motifs dans les documents XML.

En effet la recherche de motif est un sujet très étudié dans les littératures pour les motifs textuels mais n'est pas plus considéré dans la structure arborescente.

On développe un outil pour mesurer la qualité des algorithmes de recherche de motif d'arbre choisis, on combine aussi ces algorithmes avec Xpath pour prouver son efficacité.

Mots–clés : recherche de motif XML, Xpath, arbre, algorithme naïf

Abstract

The aim of this work is the study of a number of algorithms to search for patterns in XML documents.

Indeed, the pattern matching is a very studied in the literature for textual reasons but is no more considered in the tree structure subject.

In this work we develop a tool for measuring the quality of search algorithms tree pattern selected, we also combine these algorithms with XPath to prove efficiency.

Keywords : Search XML patter , Xpath, tree , naive algorithm

الملخص

يهدف هذا العمل الى دراسة عدد من خوارزميات البحث عن الأنماط في وثائق XML

في واقع الأمر البحث عن الأنماط هو موضوع جد مدروس في الأدبيات من أجل الأنماط النصية ولكنه ليس معتبرا في البنيات الشجرية

نقوم في هذا العمل بتطور أداة لقياس جودة الخوارزميات المنتقاة من خلال دمج الخوارزميات مع Xpath

لإ ثبات نجاعته

كلمات مفاتيح : البحث عن الأنماط ، XML، Xpath،شجرة ،الخوارزمية الساذجة

Sommaire

Liste des figures

Liste des tableaux

Introduction générale

Dans ces dernières années XML (eXtensible Markup Language) a joué un rôle important dans l'échange d'information sur le web et entre les applications et même dans le stockage de données. Avec le développement des outils de langage XML (XML schéma, Query, xpath ..etc.) Les systèmes d'informations utilisent de plus en plus XML pour stocker les données. Dans une entreprise l'évolution de la quantité des données de base de données XML exige l'amélioration de la performance d'interrogation de cette base.les grandes entreprises (surtout commerciales) ont un besoin major d'un système décisionnel qui aide les gérants, à comprendre le marché et l'intérêt général des clients, pour prendre la décision rentable et bénéfique pour l'entreprise.

Avec l'explosion des données souvent représentées par des documents XML sur le net, un nouveau besoin est apparu : La recherche dans les documents XML. En effet, ce terme est ambiguë et peut dire plusieurs choses, rechercher des éléments, des valeurs, des index et bien sur des motifs.

La recherche des motifs est bien présente dans la littérature et constitue un sujet d'actualité pour les chercheurs et les industriels. Plusieurs fameux travaux ont vu le jour comme les techniques des n-grammes ou le célèbre grep.

La recherche de motif est utilisée dans le processus de construction d'un système décisionnel pour choisir le meilleur index (l'index des attributs) pour l'ensemble des requêtes de ce système. Les systèmes décisionnels sont instaurés sur des requêtes complexes, dans les bases de données XML, les outils d'interrogation (xquery et xpath) parfois ne fonctionnent pas à cause de la grande quantité des données et à cause de la difficulté d'exprimer ce qu'on cherche. La recherche de motif XML est utilisée ici pour calculer l'occurrence d'un motif XML de requête, dans le but de diminuer la répétition de la demande et diminuer le temps pour la demande de ce motif .De manière générale une requête XML représente un motif XML ,donc, la recherche de motif XML est utilisée aussi pour jouer le rôle d'une requête.

Mais, XML ne suit pas la philosophie des motifs classique car sa structure est souvent vue comme arborescente plus qu'une séquence textuelle et d'ici vient le besoin de considérer les motifs d'arbres.

Heureusement, ce terme est déjà déployé dans l'informatique théorique et plus exactement dans la théorie de la structure arborescente. Plusieurs algorithmes de recherche dans les structures arborescentes existent mais la plupart dans la théorie. Une tendance vers l'application de ces techniques commence à voir le jour dans plusieurs domaines comme la bio-informatique ou les modèles de traitements d'ADN probabilisé (sont vues comme des arbres), les compilateurs et les réécritures de termes, la vérification des circuits électroniques et bien sur la recherche des motifs dans les documents XML comme un point traité par ce mémoire.

Vu l'utilité et la forte demande de la recherche de motif XML, nous lui attachons beaucoup d'intérêt. XML est une méthode pour mettre des données structurées dans un fichier texte. De ce niveau là on peut utiliser les algorithmes et les méthodes de recherche de motif de mots pour faire la recherche de motif XML. Mais notre objectif ce n'est pas cela, XML est textuel oui, mais n'est pas fait pour être lu tel qu'il est, car les données XML sont stockées en mémoire avec une structure hiérarchique (arbre).Une base de données XML représente un arbre en mémoire, les nœuds de cet arbre représente les éléments XML du fichier texte XML.Une requête XML représente un motif d'arbre XML.

La Problématique de ce mémoire est la recherche de motif XML. C'est-a-dire, j'ai un document XML. Je cherche si un motif XML « A » existe ou non dans le contenu de ce document? Le document et le motif ici représentent un arbre motif et un arbre objet.

Le **plan de travail** réalisé pour ce thème est exécuté en respectant l'équilibre entre le fond théorique (recherche de motif) et le fond pratique et technique (contexte XML) .Ce qui permet de diviser ce mémoire en trois chapitres et une annexe :

- Le premier chapitre est conçu pour encadrer le contexte XML en citant et résumant les outils et les connaissances qui sont en relations avec les procédés et les structures de XML.

- Le deuxième chapitre est conçu pour encadrer le fond théorique de la recherche de motif ainsi que les algorithmes les plus intéressants pour résoudre ce problème.
- Le troisième chapitre est conçu pour présenter les analyses, les preuves et les résultats pratiques obtenus concernant le thème étudié.
- L'annexe comporte le Model Objet Document (DOM) qui est utilisé pour la programmation XML.

Remarque : on a utilisé comme abréviations, la recherche de motif(RM), la recherche de motif XML(RMXML), la recherche de motif d'arbre(RMA) et la recherche de motif de mots (RMM)

Chapitre I : Généralités sur XML

Introduction

Notre objectif dans ce présent chapitre n'est pas l'étude complète de XML, mais c'est de toucher la structure de base et le fonctionnement général du XML. Vu la multitude des documents qui expriment les fonctionnalités et les outils fournis par le langage XML. Pour apprendre XML ou chercher n'importe quelle information en son sein, il suffit juste de taper « XML » sur le net pour découvrir son importance. Les documents qui existent en grande quantité expliquent l'immense utilisation de ce standard au niveau mondial.

Dans ce chapitre, notre étude tente de découvrir les mécanismes, les aspects et les concepts de XML qui nous aident à voir bien la problématique. Il faut évidement la reconnaissance de la structure de XML et les outils de base de ce langage. Donc nous allons essayer de faire un résumé sur le fonctionnement général de XML.

Le chapitre comporte la définition et l'historique de XML, la structure (forme textuelle, forme arborescente), les outils de définition de structure de XML (DTD, XML schéma), ainsi que la présentation des outils d'interrogation et de transformation du XML en concentrant l'étude sur l'outil Xpath sans oublier de parler du rôle de XQuery et XSLT.

1. Définition de XML

XML (*Extensible Markup Language*) en français (Langage Extensible de Balisage), est un langage de balisage pour les documents contenant des informations structurées. Un langage de balisage est un mécanisme pour identifier les structures dans un document. Aujourd'hui XML est utilisé partout en informatique pour stoker et structurer les informations, échanger des informations entre les applications… etc. Avec XML les documents de structure riche peuvent figurer en web, XML est utilisé sur le Web pour donner un contenu structuré. XML est en faite le métalangage de représentations des informations.il admet deux types de représentation, une représentation appeler forme textuelle qui contient des balises (ouvrante et fermante) et une représentation arborescente qui néglige la forme textuelle est sert à définir les informations en utilisant la structure des arbres.

2. Historique

XML ,c'est l'un des langages de balise, c'est une drivée des autres langages de balise, la notation XML, c'est une idée qui a été proposé par **James Clark** (né le 23 février 1964 à Londres).l'origine de langage XML ,c'est le « langage SGML qui a été introduit en 1986 par C. Goldfarb. SGML a été conçu pour des documentations techniques de grande ampleur. Sa grande complexité a freiné son utilisation en dehors des projets de grande envergure. En 1991, T. Berners-Lee a défini le langage HTML pour le WEB. Ce langage est une version simplifiée à l'extrême de SGML, destinée à une utilisation très ciblée. XML est, en quelque sorte, intermédiaire entre SGML et HTML. Il évite les aspects les plus complexes de SGML tout en gardant suffisamment de souplesse pour une utilisation généraliste. La version 1.0 de XML a été publiée en 1998 par le consortium W3C (World Wide Web Consortium). Une redéfinition XHTML de HTML 4.0 à travers XML a été donnée en 1999. » [1]

3. La structure de XML

3.1. La forme textuelle

La représentation d'un document XML sous la forme textuelle [1] [2] [3] est définie en utilisant les balises ouvrantes et les balises fermantes, entre ces deux on trouve les informations qui doivent être stockées ou échangées. Voir : ci-dessous un exemple de document XML

```
< ?xml version= »1.0" encoding="ISO-8859-1"?>
<!-- La ligne ci-dessus est le prologue -->
<!—Élément racine -->
<biblio>
 <!-- Premier enfant -->
 <livre>
  <!-- Élément enfant titre -->
  <titre>Les Misérables</titre>
  <auteur>Victor Hugo</auteur>
  <nb_tomes>3</nb_tomes>
 </livre>
```

```
 <livre>
  <titre>L'Assomoir</titre>
  <auteur>Émile Zola</auteur>
 </livre>
<!--lang c'est un attribut de l'element livre  -->

 <livre lang="en">
  <titre>David Copperfield</titre>
  <auteur>Charles Dickens</auteur>
  <nb_tomes>3</nb_tomes>
 </livre>
</biblio>
```

Un document XML est composé de deux parties, une partie nommée « le prologue » et une partie nommée « arbre d'éléments » qui commence toujours par l'élément racine.

Dans le **prologue** on trouve deux sous-parties :

La première sous-partie de prologue contient les instructions de traitement, voir ci-dessous

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

Dans cette instruction on trouve, l'attribut **version** pour préciser la version de XML utilisée, dans le document précédent, la version est 1.0.L 'attribut **encoding**, c'est le jeu de codage de caractères utilisés. Dans le document Le jeu de caractères utilisé «*ISO-8859-1 »,* adapté pour la langue française. L'attribut **Standalone**, c'est pour indiquer la dépendance du document par rapport à un document. Par défaut, La valeur **standalone** est « yes » qui signifie que le processeur de l'application n'attend aucune déclaration de type de document extérieur au document.

La deuxième sous-partie de prologue, c'est pour la Déclaration de type de document (DTD), la DTD est un outil utilisé pour définir la structure de document XML, elle est présente dans la partie de prologue avec deux façons externe ou interne voir ci-dessous :

Externe

```
<!DOCTYPE biblio SYSTEM "biblio.dtd">
```

Interne

```
<!DOCTYPE biblio[
<!ELEMENT biblio (livre)*>
<!ELEMENT livre (titre, auteur, nb_pages)>
 <!ATTLIST livre
  type (roman | nouvelles | poemes | théâtre) #IMPLIED
  lang CDATA "fr"
 >
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT nb_pages (#PCDATA)>
]>
```

Remarque : l'emplacement de DTD est juste après la partie des instructions de traitement. Il y a d'autres outils pour la déclaration de structure de document XML mais il sont utilisé avec d'autres façons.

Dans la partie de **prologue** on trouve, aussi, les commentaires qui peuvent exister aussi dans la partie d'arbre élément. Généralement un commentaire est utilisé pour expliquer une partie du code (une instruction, des éléments….etc.), les commentaires XML, sont comme tout commentaire dans un langage de programmation. La syntaxe utilisée pour écrire un commentaire XML, c'est comme la syntaxe du langage HTML, voir ci-dessous

```
<!-- ceci est correct -->
```

L'arbre d'élément, c'est la constitution d'éléments imbriqués les uns dans les autres (parents-fils), l'arbre d'élément commence toujours par l'élément racine, voir l'exemple (1). L'arbre élément c'est l'élément racine avec tous ses fils. Les éléments peuvent contenir des attributs, voir l'exemple(1). L'élément livre admet un attribut appeler « lang »a la valeur « en », un élément peut admettre plusieurs attributs de types différents (chaine de caractère, entier,….etc.).Le contenu d'un élément peut être un type simple (texte, nombre…etc.) ou un

élément fils (le contenu c'est un arbre d'élément).Il y a une possibilité de trouver un élément vide qui n'admet aucun contenu.

3.2. La forme arborescente

La forme arborescente (arbre XML) [2] est la représentation de base d'un document XML, parce que la forme textuelle est juste la description de cette forme. La forme textuelle est utilisée pour le stockage et l'échange de données, par contre la forme arborescente est utilisée pour faire la mise-à-jour (exécuter les opérations de modification) des documents XML. La forme arborescente donne une meilleur manipulation des données car la manipulation des éléments de l'arbre XML se fait hiérarchiquement c'est-à-dire, dans la forme textuelle lorsque on fait la recherche des éléments par exemple, notre recherche se fait avec une manière séquentielle parce que cette dernière est une lecture de l'arbre XML qui admet un sens de lecture dans l'arbre XML. Mais lorsque on fait la recherche dans la forme arborescente, elle se fait dans l'arbre lui-même (tout les sens sont possibles). On déduit que la forme textuelle est la traduction de l'arbre XML. Comme tout arbre, un arbre XML est une structure de donnée riche. En outre, certaines spécificités le rendent encore plus riche. En effet, il est composé de nœuds interconnectés avec des relations parent-fils, la richesse de l'arbre XML vient de la présence de plusieurs type de nœuds : le nœud racine qui représente le nœud de type document, il y a aussi les nœuds de type éléments, les nœuds de type attribut, texte, instruction de traitement, commentaire. Voici ci-dessous un exemple d'arbre XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!— le code -->
<A>bonjour A
<B>bonjour B</B>
<D attr1="1" attr2="ALI">
<C/>
</D>
<![CDATA[12546687 and 125]]>
</A>
```

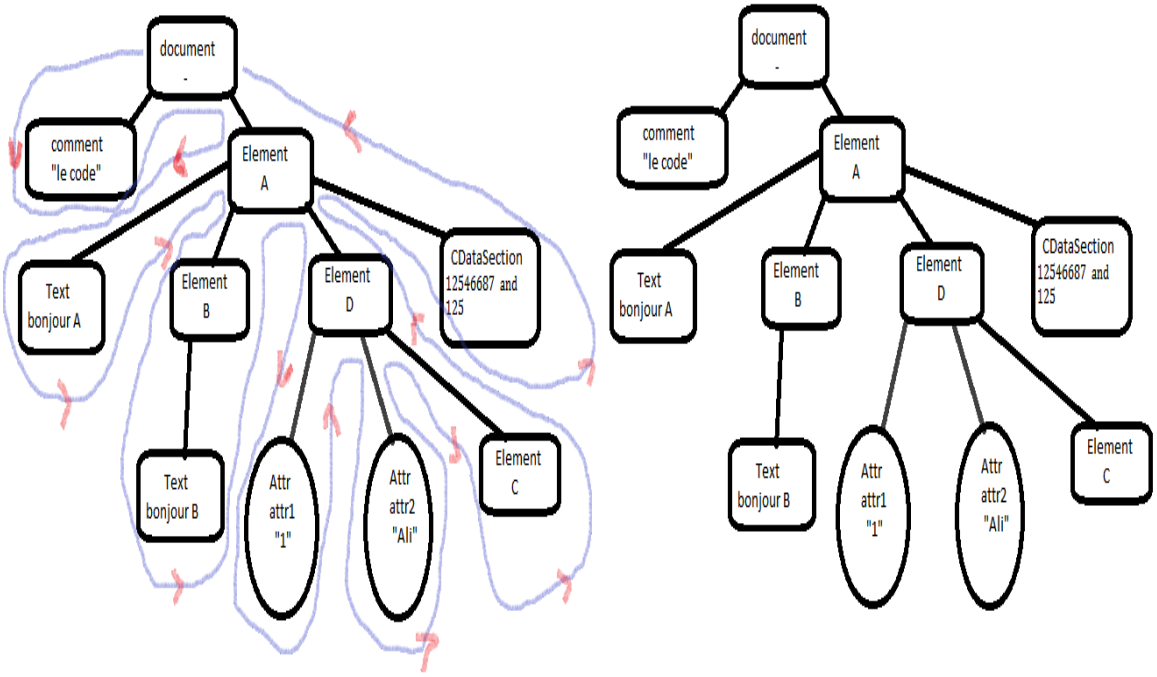


Figure I.1 la forme arborescente de XML

Si on regarde bien l'arbre XML la forme textuelle est obtenue avec l'application d'un parcours qui lire chaque fois un composant de l'arbre XML

4. Les outils de définition de structure XML

Dans cette partie, on parle de la description de structure d'un document XML. Pour chaque outils de description de structure d'un document XML (dtd, XML shéma), on essaye de faire un résumé comment faire la description d'un document XML.

4.1. Déclarations de Type de Document (DTD)

Il y a deux façons pour déclarer le type d'un document XML (voir la partie DTD de [3] et de [4]), comme on a déjà vu interne ou externe. Un DTD, commence toujours par l'instruction « ! DOCTYPE » et l'élément racine du document XML voir ci-dessous :

<!DOCTYPE l'élément racine[le contenu (les déclarations des éléments)]>

La déclaration des éléments dans les DTD commence toujours par « < ! element » avec le nom et le type d'élément (il y a plusieurs types) est terminé par « > » comme ci-dessus

<!ELEMENT nom_element type_element>

Il y a plusieurs types d'élément, le tableau ci-dessous représente les différents types de contenu possible pour un élément

Le type	L'instruction	exemple	Contre exemple
Contenu libre	**<!ELEMENT doc ANY>**	**<doc>** **Ceci est une documentation** **très simple.** **</doc>** **Ou** **<doc>** **** **documentation** **** **** **Importante** **** **</doc>.** **Ou** **<doc />**	

Contenu vide	**<!ELEMENT toc EMPTY>**	**<toc></toc>** **Ou** **<toc/>**	**<toc> </toc>** **L'espace « «**
Contenu textuel	**<!ELEMENT comment (#PCDATA)>**	**<comment>** **<!-- no comment -->** **</comment>** **Si possible de trouvé** **<comment/>**	**<comment>** **<element/>** **</comment>**
Contenu composé	**<!ELEMENT produit (nom,prix,comment)>** **<!ELEMENT nom (#PCDATA)>** **<!ELEMENT prix (#PCDATA)>** **<!ELEMENT comment (#PCDATA)>**	**<produit>** **<nom>** **XML par la pratique** **</nom>** **<prix>20</prix>** **<Comment>** **programmer des applications XML.** **<Comment>** **</produit>**	**<produit>** **<prix>20</prix>** **<Comment>** **programmer des applications XML.** **<Comment>** **</produit>**

Tableau I.1 : les différents types de contenu possible d'élément en DTD

En plus de cela, en DTD on peut gérer les répétions de contenu d'un élément par l'ajout de l'un des caractères « ? », « + », « * » après les éléments, on peut aussi faire l'alternative entre les fils d'un élément par « | » ou la séquence entre les éléments par «, », voir ci-dessous

<!ELEMENT **chapitre (nom, date?, auteur*, intro?,(nom-de-section,corps-de-section)+)>**	**<chapitre>** **<nom>Utiliser les DTD</nom>** **<date>10/11/2002</date> <!-- optionnel -->** **<!-- 0 à n fois -->** **<auteur>…</auteur><auteur>…</auteur>** **<!-- optionnel -->** **<intro>…</intro>** **<!-- 1 à n fois -->** **<nom-de-section>Préambule</nom-de-section>** **<corps-de-section>…</corps-de-section>** **</chapitre>**

On passe à la déclaration des attributs. Un **attribut**, c'est un champ à remplir, il admet un type et par fois une valeur ou liste des valeurs. La déclaration des attributs est écrite après la déclaration de l'élément. Voir comment déclarer un attribut ci-dessous.

```
<!ATTLIST  nom_élément
   nom_attribut_1  type_attribut_1  déclar_de_défaut
   nom_attribut_2  type_attribut_2  déclar_de_défaut
   ...      >
```

Voici quelque instruction utilisée pour la déclaration des attributs dans le tableau ci-dessous

'valeur'	**valeur par défaut**
#REQUIRED	**l'attribut doit être renseigné**
#IMPLIED	**l'attribut est facultatif**
#FIXED 'valeur'	**l'attribut a toujours la même valeur**
CDATA	**le type de l'attribut est texte**

Tableau I.2 : les instructions de déclaration de type d'attribut en DTD

Avec toutes ces options la DTD admet des limites, parmi ces limites. Les DTD ne sont pas au format XML, il est nécessaire d'utiliser un outil spécial pour manipuler ces fichiers, les DTD ne supportent pas les « espaces de nom » (après nous expliquons cette notion), pratiquement il n'est pas possible d'importer des définitions de balises définies par ailleurs dans un fichier XML défini par une DTD.

4.2. XML schéma (XSD)

XML schéma (voir la partie Initiation aux Schéma XML [4] et [5]), c'est un outil de définition de structure du document XML, il est comme outil DTD, la différence entre DTD et XML schéma c'est la syntaxe, Un fichier XML schéma admet la structure de document XML (admet la syntaxe XML), c'est-à-dire il doit contenir un prologue et un arbre d'élément .Au niveau de fonctionnalités XML Schéma propose des nouveautés en plus des fonctionnalités fournies par les DTD :

- Le typage des données est introduit, ce qui permet la gestion de booléens, d'entiers, d'intervalles de temps... Il est même possible de créer de nouveaux types à partir de types existants.
- La notion d'héritage. Les éléments peuvent hériter du contenu et des attributs d'un autre élément.
- Support les espaces de nom.
- Les indicateurs d'occurrences des éléments peuvent être tout nombre non négatif (rappel : dans une DTD, on est limité à 0, 1 ou un nombre infini d'occurrences pour un élément).
- Les schémas sont très facilement concevables par modules.

Un **élément** dans un fichier XML schéma, se déclare avec la balise <xsd:element> voir ci-dessous comment déclarer un élément.

```
<xsd:element name="le nom-element " type="type-element" />
```

L'emplacement de la déclaration des éléments est entre les balises de la racine de XML schéma (racine d'un XSD : < schema> ... </schema>), chaque élément de XML schéma doit

avoir un nom et un type, le nom de l'élément est décrit à l'aide de l'attribut « name », le type d'élément est déclaré à l'aide de l'attribut type. Le type de l'élément en XML schéma est lié toujours avec la forme de l'élément, lorsque L'élément contient des éléments-enfants ou possèdent des attributs, le type des éléments sera un type *complexe*, mais lorsque les éléments ne contiennent pas le type des éléments sera un type *simple*. Vous trouvez les déclarations des types des éléments par défaut après les déclarations des éléments.

Un **attribut** XML schéma c'est comme l'attribut en DTD, il admet un nom et un type, l'attribut XML schéma admet comme type l'un des types simple. L'attribut en XML schéma ne doit pas contenir des éléments ou des attributs. La déclaration d'un attribut appartient à la définition de type complexe, donc vous trouvez la déclaration de l'attribut après l'un des éléments **xsd:sequence**, **xsd:choice** et **xsd:all** (ces trois éléments servent à gérer l'alternative et la séquence des éléments ils sont appelés les indicateurs d'ordre) .voici comment déclarer un élément et l'emplacement de la déclaration voir ci-dessous.

```
<xsd:attribute name="nom-attribut" type="type-attribut" />
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="contacts" type="typeContacts" />
 <xsd:element name="remarque" type="xsd:string">    <!-- déclarations de types ici -->
  <xsd:complexType>
  <!-- déclarations du modèle de contenu ici -->
   <xsd:attribute name="maj" type="xsd:date" />
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

L'attribut d'un XML schéma peut avoir trois attributs optionnels : use, default et fixed. On utilise ces trois attributs pour paramétrer ce qui est acceptable ou non dans le fichier XML final (attribut obligatoire, optionnel, possédant une valeur par défaut...).voir ci-dessous

```
<xsd:attribute name="date" type="xsd:date" use="optional" default="2003-10-11" />
```

L'attribut date est optionnel, la valeur par défaut de cet attribut est 11 octobre 2003. Voir ci-dessous comment déclarer un élément qui contient un texte et un attribut

```
<xsd:element name="elt">
 <xsd:complexType mixed="true">
  <xsd:attribute name="attr" type="xsd:string" use="optional" />
 </xsd:complexType>
</xsd:element>
```

Il faut juste charger l'attribut « mixed » de l'élément type complexe par la valeur « true » parce qu'il est par défaut false.

Il y a deux types de données, les **types de données simples** (float, integer,….) qui sont les types prédéfinis dans la bibliothèque de types intégrés de XML Schéma et les **types de données complexes** qui seront a déclaré des structures d'éléments XML c'est-a-dire, dans un type complexe on peut trouver des éléments et des attributs. Un élément à un type complexe c'est un élément qui contient des éléments et des attributs définis par ce type, un élément a un type complexe c'est un élément vide ou un élément qui contient d'autres éléments ou un élément qui contient seulement du texte, ou un élément qui contient d'autres éléments et du texte. Dans la déclaration des types de données complexes on utilise, les indicateurs d'ordre et les indicateurs d'occurrence et les indicateurs de groupes. Pour contrôler l'usage de l'élément à l'intérieur du type complexe, voir ci-dessous.

Les indicateurs	**Le nom de l'indicateur**	**Comment utiliser l'indicateur**	**Explication**
D'ordre	All	<xs:complexType> <xs:all> <xs:element name="firstname" type="xs:string"/> <xs:element name="lastname" type="xs:string"/> </xs:all> </xs:complexType>	les éléments peuvent apparaître dans n'importe quel ordre et chaque élément doit se produire une et une seule fois
	Choice	<xs:complexType> <xs:choice> <xs:element name="employee" type="employee"/> <xs:element name="member" type="member"/> </xs:choice> </xs:complexType>	un seul élément parmi les éléments peut se produire
	sequence	<xs:complexType> <xs:sequence> <xs:element name="firstname" type="xs:string"/> <xs:element name="lastname" type="xs:string"/> </xs:sequence> </xs:complexType>	les éléments doivent apparaître dans un certain ordre
D'occurrence	maxOccurs	<xs:complexType> <xs:sequence> <xs:element name="full_name"	le nombre maximum de fois de l'élément « child-name » peut se produire 10 fois, par défaut au minimum égal 1

		type="xs:string"/> <xs:element name="child_name" type="xs:string" maxOccurs="10"/> </xs:sequence> </xs:complexType>	
	minOccurs	<xs:complexType> <xs:sequence> <xs:element name="full_name" type="xs:string"/> <xs:element name="child_name" type="xs:string" maxOccurs="10" minOccurs="0"/> </xs:sequence> </xs:complexType>	le nombre minimum de fois de l'élément « child-name » peut se produire 0 fois
De groupe	Group name	<xs:group name="persongroup"> <xs:sequence> <xs:element name="firstname" type="xs:string"/> <xs:element name="lastname" type="xs:string"/> <xs:element name="birthday" type="xs:date"/> </xs:sequence> </xs:group>	Pour faire l'appel à ce groupe d'éléments utilise <xs:group ref="persongroup"/>
	Attribute Group name	<xs:attributeGroup name="personattrgroup"> <xs:attribute name="firstname" type="xs:string"/> <xs:attribute name="lastname"	Pour faire l'appel à ce groupe d'attribut utilise <xs:attributeGroup ref="personattrgroup"/>

		type="xs:string"/> <xs:attribute name="birthday" type="xs:date"/> </xs:attributeGroup>	

Tableau I.3 : Les indicateurs de XSD

On a cité avant que parmi les différences entre XML schéma et la DTD c'est utilisation de l'espace de nom. On utilise **les espaces de nom** en XML schéma pour distinguer les éléments et attributs de différente applications XML qui ont le même nom et aussi pour grouper les éléments et attributs d'une même application XML pour être reconnu tout simplement, « Un espace de noms XML est une recommandation du W3C qui permet d'employer des éléments et des attributs nommés dans une instance XML. Une instance XML peut contenir des noms d'éléments ou d'attributs de plus d'un vocabulaire XML. Si on attribue à chaque vocabulaire un espace de noms, alors on peut résoudre les ambiguïtés entre des noms identiques d'éléments ou d'attributs. Les noms d'élément au sein d'un même espace de noms doivent être uniques. » [6], Voici un exemple qui exprime dans quel cas on utilise les espace de nom « Prenons comme exemple une instance XML qui contiendrait les références à un client et un produit commandé. L'élément client et l'élément produit pourraient avoir un élément fils "ID_number". Les références à l'élément ID_number seraient alors ambigües, à moins que les deux éléments de noms identiques mais différents du point de vue sémantique aient été associés à des espaces de noms qui permettent de les différencier ; par exemple cl:ID_number et pr:ID_number. » [6]

5. Les outils de transformation et d'interrogation de XML

Pour ce point il y a trois outils principaux, lorsque les documents de XML sont volumineux, il faut un outil pour naviguer (la sélection des parties) dans ces documents XML, Il faut aussi des outils pour faire les calculs et la transformation de ce document(les données sont stockées pour que après doit être analysé simplement). lorsque j'ai un document XML qui admet plusieurs balises semblables l'analyse est difficile, donc il faut utiliser un outils pour faire la sélection de partie de document et le calcul dans ces parties(les requêtes XML) ,il faut aussi utiliser un outil de transformation de ce document pour charger les données de

document XML ou les résultats de calcul , dans une interface significative(une vision claire des données) .

« **XSLT** (*eXtensible Stylesheet Language Transformations*), défini au sein de la recommandation XSL du W3C, est un langage de transformation XML de type fonctionnel. Il permet notamment de transformer un document XML dans un autre format, tel PDF ou encore HTML pour être affiché comme une page web. » [7]

« **XQuery** est un langage de requête informatique permettant non seulement d'extraire des informations d'un document XML, ou d'une collection de documents XML, mais également d'effectuer des calculs complexes à partir des informations extraites et de reconstruire de nouveaux documents ou fragments XML. XQuery est une spécification du W3C dont la version 1.0 finale date de Janvier 2007, et dont l'élaboration a demandé près de huit années. XQuery a été développé conjointement avec XSLT 2, une révision majeure du langage de transformation XML XSLT, avec lequel il partage le sous-ensemble XPath 2. » [8]

« **XPath** est un langage (non XML) pour localiser une portion d'un document XML. Initialement créé pour fournir une syntaxe et une sémantique aux fonctions communes à XPointer et XSL, XPath a rapidement été adopté par les développeurs comme langage d'interrogation simple d'emploi. » [9]

D’après ces définitions, XSLT ajoué le rôle de transformateur de document XML de forme textuelle vers une autre forme simple à lire (HTML ,PDF …etc). XQuery c’est est un standard du W3C ,il est comme tout langage de requête par exemple SQL, en général XQuery fait l’analyse des documents XML, l’analyse comporte des calculs complexes par exemple l’intersection entre deux documents XML union…etc. .XSLT utilise XQuery car le résultat d’une requête XQuery par fois représente un arbre XML(document XML) donc il faut utiliser l’outils de transformation XSLT pour que les résultats doivent être clair et doivent être obtenu par une méthode procédurale .l’intersection entre XSLT et l’XQuery c’est l’utilisation des expressions Xpath . Xpath est langage de requête qui fait la localisations des nœuds de l’arbre XML par utilisation des expressions de chemin, pour naviguer dans le document XML , d’autre part XPath est aussi une recommandation W3C.

Le principe de fonctionnement de l'Xpath c'est de considérer l'expression de l'Xpath (\a\b\t) comme un ensemble d'étapes (\etape1\etape2\.....\etapeN), chaque étape successive détermine un ensemble de nœuds à partir de nœud contexte, c'est-a-dire chaque étape utilise les nœuds sélectionnés par l'étape précédente ainsi de suite. Entre chaque deux étape ou chaque deux nœud on trouve des axes, dans les expressions Xpath il y a 12 types d'axes voir ci-dessous [10] :

– child (axe par défaut): enfants directs du nœud contexte

– parent (..): nœud parent

– attribute (@): nœuds attribut du nœud contexte

– descendant (//): nœuds descendants du nœud contexte

– descendant-or-self: descendants, y compris le nœud contexte

– ancestor: nœuds ancêtres du nœud contexte

– ancestor-or-self: ancêtres, y compris le nœud contexte

– following: nœuds suivants dans l'ordre du document

– following-sibling: frères suivants dans l'ordre du document

– preceding: nœuds précédents dans l'ordre du document

– preceding-sibling: frères précédents dans l'ordre du document

– self(.): le nœud contexte lui-même

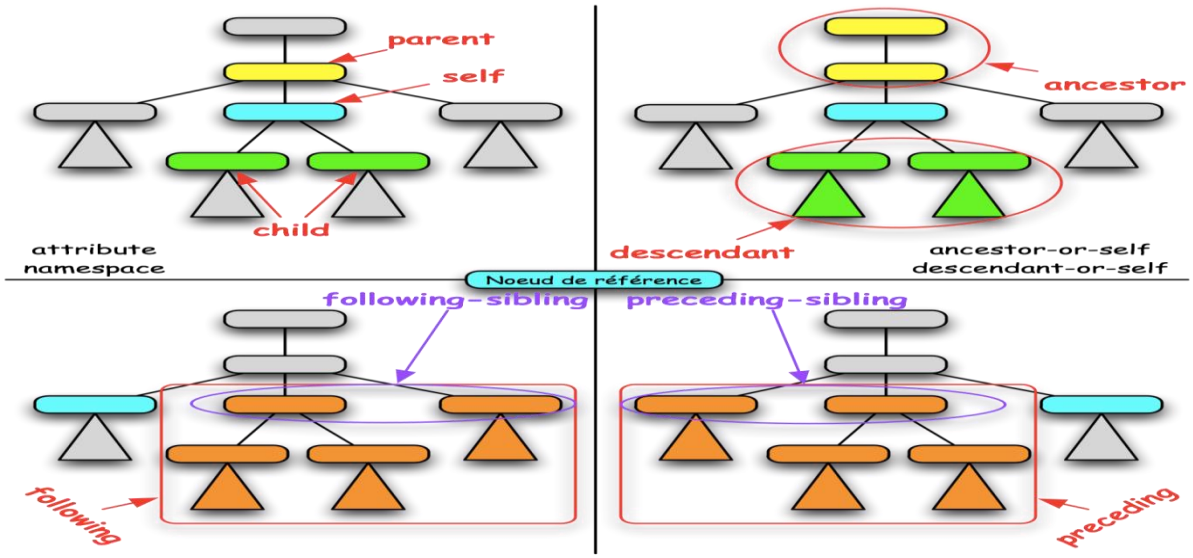


Figure I.2 : les différents axes de l'Xpath

Les requêtes Xpath sont divisées en trois, il y a les requêtes de recherche simple, les requêtes de recherche par contexte et les requêtes de recherche par prédicat, pour comprendre ces trois voici ci-dessous un document XML et des exemples de requêtes Xpath à exécuter sur ce document

```
<?xml version="1.0" encoding="UTF-8"?>
<a>
  <b  nom='fr' >
    <c>Bonjour</c>
    <d> </d>
    <k>texte1</k>
  </b>
  <b att='en'>
     <c>Hello</c>
     <c>bonjour</c>
     <k>texte2</k>
  </b>
  <d id='12'>
    <g>
      <h> </h>
    </g>
  </d>
  <d id='10'>  </d>
    <ee>texte1  </ee>
    <de>texte2  </de>
</a>
```

Les requêtes	**L'expression**	**Le rôle**	**exemple**	**Le résultat**
Recherche simple	/a/b	Liste des éléments dont le chemin dans l'arborescence correspond à la requête	/a/b/c	<c>Bonjour</c> <c>Hello</c> <c>bonjour</c>
	//	On ne tient pas en compte la profondeur de l'élément dans l'arborescence	//c	<c>Bonjour</c> <c>Hello</c> <c>bonjour</c>
	|	C'est la composition de requête	//c | //k	<c>Bonjour</c> <k>texte1</k> <c>Hello</c> <c>bonjour</c> <k>texte2</k>
	*	Tous les éléments fils de la sélection	//b/*	<c>Bonjour</c> <d> </d> <k>texte1</k> <c>Hello</c> <c>bonjour</c> <k>texte2</k>
Recherche par contexte	descendant	Fils d'un nœud et leurs descendants	/a/d/descendant ::*	<g> <h> </h> </g>

	Ancestor	Père d'un nœud et ses ancêtres	//g/ancestor ::*	Le résultat c'est tout l'arbre XML
	Child	Fils d'un nœud	/a/d/child ::*	<g> <h> </h> </g>
	Attribute(@)	Pour séléction l'attribut d'un nœud	//d/attribute ::*	id='12' id='10'
	Parent(..)	Parent d'un nœud	/a/d/g/h/parent ::*	<g> <h> </h> </g>
Recherche par prédicat	[j]	Le Jème élément de la sélection	/a/b[2]	 <c>Hello</c> <c>bonjour</c> <k>texte2</k>
	[@ att= 'val']	Elément dont l'attribut att a pour valeur val	/a/b[@att='en']	 <c>Hello</c> <c>bonjour</c> <k>texte2</k>
	text()	Tous les nœuds enfants de type textuels	//c[2]/text()	bonjour
	last()	Dernier élément de la sélection	//b[last()]	 <c>Hello</c> <c>bonjour</c> <k>texte2</k>

	Position()	Renvoi l'index de position du nœud relativement au noued	//b[position()=2]	 <c>Hello</c> <c>bonjour</c> <k>texte2</k>
	Name()	Nom de l'élément	//*[name()='k']	<k>texte1</k> <k>texte2</k>
	contains()	Condition sur les chaines de caractère	//*[contains(name(),'E')]	<de>texte2 </de>
	count()	Compte les éléments de la sélection	//*[count(*)=2]	Sélectionné tous les éléments qui n'admettent pas des fils

Tableau I.4 les différentes instructions de l'Xpath

Conclusion

Selon mon modeste avis, XML c'est juste une structure de stockage et de traitement de données. Lorsqu'on veut stocker ou transférer les données, il faut utiliser la forme textuelle parce qu'elle est séquentielle ce qui implique qu'elle est stockable et transférable .Mais lorsqu'on veut traiter les données il est préférable d'utiliser la forme arborescente, car cette forme expose les éléments de document XML en une forme hiérarchique ce qui permet d'optimiser le cout des opérations de mise-à-jour. Les outils de recherche sur XML fonctionnent avec la forme arborescente de XML. Nous nous intéressons à la recherche dans l'arbre XML et non dans le document XML (forme textuelle). Alors il faut étudier les différents algorithmes de RM pour voir le chemin menant à la RMXML.

Chapitre II : Etat de l'art

Introduction

Dans le domaine de stockage de données on trouve plusieurs structures de données, la structure arborescente (arbre) c'est l'une de ces structures .Il y a plusieurs travaux liés à cette structure, parmi ces travaux la recherche de motif d'arbre(RMA). La RMXML est étroitement liée à la RM dans une structure arborescente car, un document XML représente au fait une extension de la structure arborescente, donc pour trouver la solution de notre problématique nous devons étudier les différents algorithmes de RMA. Il est important de noter que les algorithmes de RMA dans la plupart du temps représentent des améliorations de la recherche RMM.

Dans ce chapitre l'étude expose la partie théorique sur les langages d'arbre inclue dans le déroulement et l'application physique des algorithmes de RMA, le problème de RMA qui représente l'origine du problème de RMXML, les algorithmes de RMM (l'algorithme d'Aho-Corasick, l'algorithme de Knuth-Morris-Pratt) qui sont utilisés pour améliorer et écrire les algorithmes de RMA (L'algorithme descendant de Hoffmann et O'Donnell (1982), L'algorithme de Ramesh et Ramakrishnan (1992) ainsi que l'algorithme naïf de RMA). A la fin de l'étude vous trouvez la partie d'analyse sur l'application des algorithmes de RMA.

1. La structure arborescente et la recherche de motif

La recherche en générale est liée toujours au domaine de recherche et à la structure de ce domaine, par exemple dans la vie courante un livre est toujours conçu avec un sommaire pour quoi ?

La réponse : le sommaire représente la structuration de livre qui permet l'amélioration des opérations d'extractions des informations, donc je veux optimiser le temps par le sommaire. On revient en informatique à la manipulation des données qui exige toujours la structure de données, comme l'exemple cité précédemment.

La structure de données, c'est la façon de stocker les données en mémoire. Une mémoire est composée de cellules. Pour préciser notre définition, une structure de données c'est une méthode de remplissage et d'extraction des données des cellules de mémoire.

Il y a plusieurs structures de données, parmi ces structures on cite : Les structures séquentielles, les structures arborescentes, les structures élémentaires….Etc. chacune a son utilisation. L'arbre c'est l'un de ces structures, dans cette structure qui s'appelle par fois structure hiérarchique on trouve plusieurs types d'arbres et pour chaque type, on trouve des avantages et des inconvénients au niveau de capacité de stockage et par rapport au temps d'exécution des opérations comme la recherche, la suppression et l'insertion …etc.

La recherche d'information dans un arbre est liée au format d'information à chercher car « Un arbre est une structure de données (souvent dynamique) représentant un ensemble de valeurs organisées hiérarchiquement. Chaque valeur est stockée dans un nœud. Les nœuds sont connectés entre eux par des relations parent/fils. A part le nœud racine, tous les autres nœuds ont exactement un seul nœud parent et zéro ou plusieurs nœuds fils. Le nœud racine n'a pas de parent et possède zéro ou plusieurs fils. Les nœuds qui n'ont pas de fils sont appelés feuilles (ou nœuds externes), les autres (ceux qui ont au moins un fils) sont appelés nœuds internes.» [11] Voir la figure (Figure II.1).

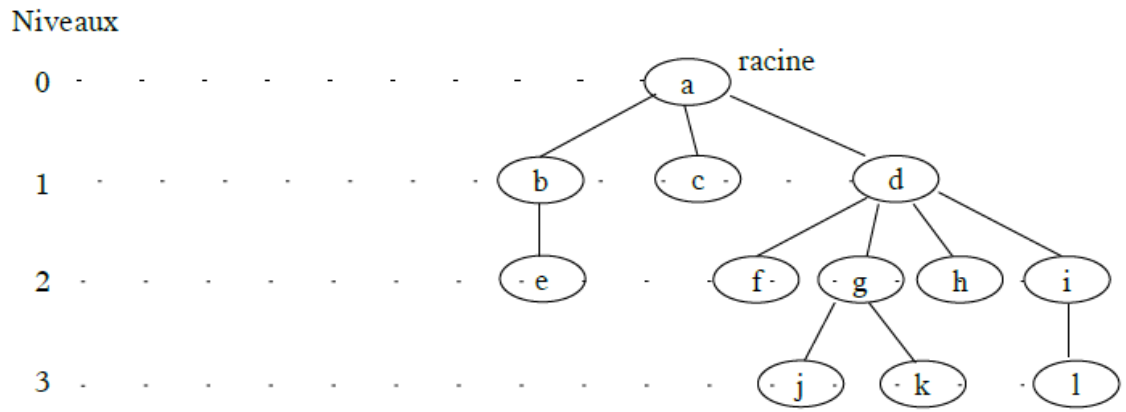


Figure II.1 : la forme d'un arbre

Remarquer : lorsque l'information existe dans un nœud, l'opération de recherche d'information, est une recherche d'un nœud dans un arbre. Mais, si l'information représente une séquence des nœuds, la recherche sera une recherche d'un arbre dans un arbre, c'est exactement la RMA.

2. Les langages réguliers d'arbre

Selon notre modeste avis les langages réguliers d'arbre représentent une science théorique. Cette théorie est comme la théorie des langages réguliers de mots.les langages réguliers d'arbre servent à définir les arbres par des représentations. Les représentations sont des automates, des expressions régulières ou des grammaires régulières. On trouve dans les langages réguliers d'arbre et de mot, des définitions et concepts partagés entre eux. Pour comprendre les définitions de cette théorie il faut avant tous comprendre les notations utilisées. Dans cette théorie on a sélectionné (avec [12]) les principes(les définitions) qui peuvent nous aider dans la RMXML

2.1. Les notations utilisées dans la théorie d'arbres

Dans la théorie des langages réguliers d'arbre vous trouvez les notations suivantes :

- Σ *:* pour l'alphabet des terminaux et *N* pour les non-terminaux
- *a, ..., e :* pour les symboles terminaux et *A, ...,E, S* pour les non-terminaux
- *G :* pour les grammaires
- *L* : pour les langages
- *i, ..., n :* pour les nombres entiers
- ε : pour dénoter le mot vide, la longueur de mots vide ε est $|\varepsilon| = 0$.

Vous trouvez d'autres symboles que ça, ils sont pour les automates d'arbre.

2.2. Les définitions et les concepts liés aux arbres

Définition1 : Un alphabet ordonné est un couple(Σ, r) , où Σ est un ensemble fini et r est une application de Σ à N. L'arité d'un symbole $a \in \Sigma$ est $r(a)$. L'ensemble de symboles d'arité p est Σ_r.

L'arité est utilisée pour décrire le nombre des nœuds fils pour un nœud quelconque a, Les éléments d'arité (par exemple r(a)) 0, 1, ..., p sont appelés respectivement constantes, Unaire, binaire, ..., p airs. On pratique (dans arbres XML par exemple) les arités sont des pointeurs qui permettent l'accès vers les fils et qui sont numéroté de (0) jusqu'à aux(le nombre de fils-1). En théorie nous devons parfois utiliser les parenthèses et les virgules Pour décrire les symboles avec arite, Par exemple la représentation en utilisant les parenthèses de l'arbre de la figure II.2 ci-dessous est $b(b, k, b(k, l(k)))$.

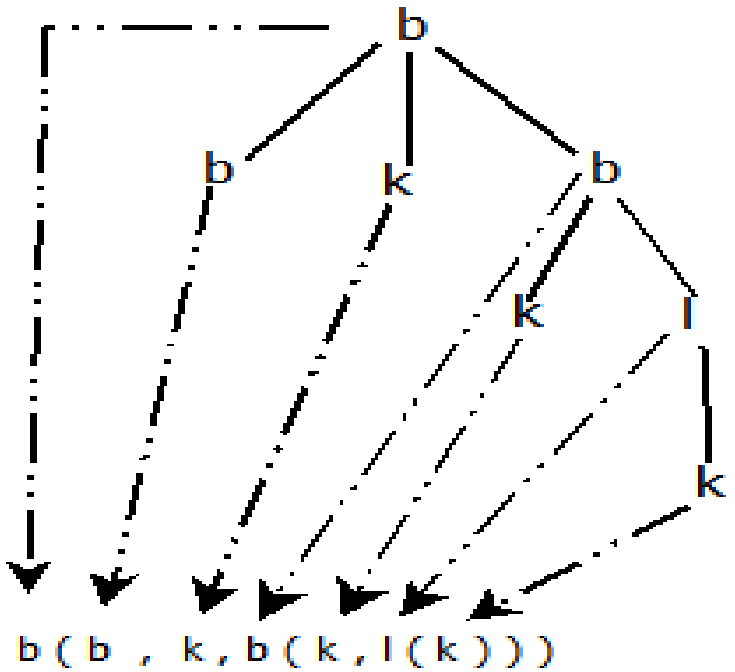


Figure II.2 : écrire un arbre avec les parenthèses

En théorie d'arbre, la définition formelle d'arbre est liée avec le domaine d'arbre qui représente la structure d'arbre ou la forme d'arbre (la caractérisation de tous les chemins de cet arbre). Le domaine d'arbre est obtenu par l'élimination des étiquètes et le replacement par des valeurs qui expriment la position de nœud par exemple, voir ci-dessous l'arbre (a) est son domaine l'arbre (b)

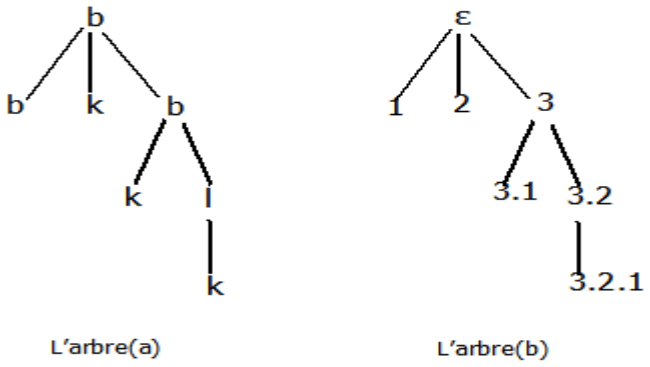


Figure II.3 : L'arbre(a) et son arbre de domaine (b)

Définition2 : La fonction **partielle** (infixe), notée t/n, est définie pour un arbre $t \in T(\Sigma, r)$ et une position $n \in D_t$ comme étant le sous arbre de t à partir du nœud n. Notons que $t/\varepsilon = t$.

La fonction partielle permet la récupération des sous-arbres en fonction du domaine de la racine(n) de sous-arbre et en fonction de l'arbre qui contient le sous-arbre lui-même, par exemple $t/3$ de l'arbre précédent de la figure ( ) $t/3 = \{(3, b), (3.1, k), (3.2, b), (3.2.1, c)\}$ Pour éclaircissent voir la figure II.4

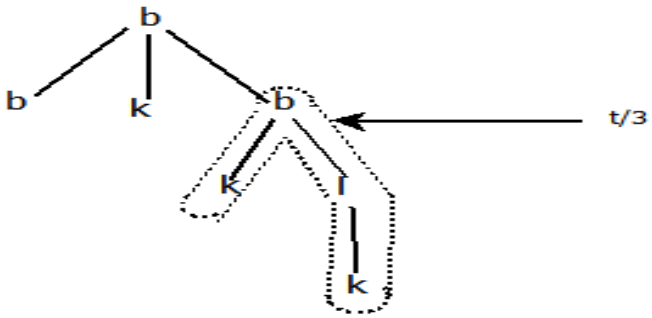


Figure II.4 : la fonction partielle d'arbre t/n

Lorsqu'on parle de la représentation de l'arbre, un arbre peut être représenté par ses chemins qui contiennent les arités. Le chemin dans un arbre représente la séquence du nœud, de la racine vers la feuille.

Définition 3 : La fonction $SPath(t)$ pour un arbre $t \in T(\Sigma, r)$ est définie par :

- $SPath(t) = \{t(\varepsilon)\}$ si t est une feuille,
- $SPath(t) = \{t(\varepsilon)\} \cup i, i = 1 \ldots r(t(\varepsilon)), \{i\}.Spath(t/i)$ sinon

Pour comprendre la définition t(ε) rendre le nœud racine, avec l'exemple précédent $t(3) = b$. Par exemple soit :

$t = a(b(c), a(c, c))$,$SPath(t) = \{a1b1c, a2a1c, a2a2c\}$ et $SPath(t/2) = \{a1c, a2c\}$.

Pour plus de détails voir la figure II.5.

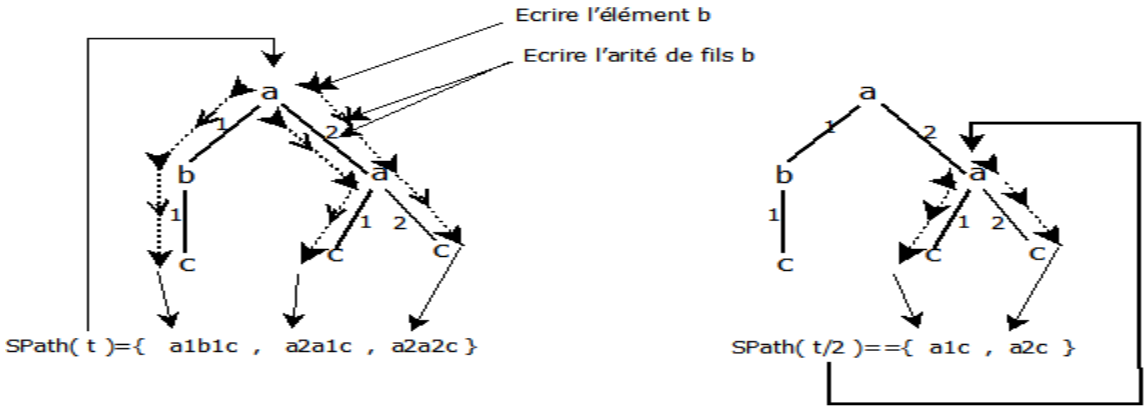


Figure II.5 : la fonction Spath pour le calcul des chemins

3. Le problème de recherche de motif d'arbre

Le problème de RMA (pattern) est étroitement lié à la définition de motif d'arbre. Un motif d'arbre [12], c'est un arbre dans lequel des nœuds variables peuvent figurer, les emplacements de ces nœuds doivent être au niveau des feuilles .les nœuds variables v de l'arbre de motif représentent parfois des sous-arbres dans l'arbre objet t .La définition formelle de motif d'arbre par [12], c'est l'ajout d'un nouveau symbole $\{v\}$ a l'ensemble d'alphabet Σ,donc il faut construire un nouveau alphabet (Σ', r'), $\Sigma' = \Sigma \cup \{v\}$, tel que $r'(v) = 0, r'(a) = r(a)$ pour tout $a \in \Sigma$ .La recherche de motifs est définie à l'aide de la fonction partielle $Match$

Définition 4 : la fonction partielle $Match \in T(\Sigma, r) \times T'(\Sigma', r') \times N_+^* \to B$ *peut être définie de manière récursive pour chaque motif* $p \in T(\Sigma', r')$ *, un arbre objet* $t \in T(\Sigma', r')$ *et* $n \in N_+^*$ *par* $Match(p, t, n) =$

$$n \in D_t \qquad si\ p = v$$

$$n \in D_t \wedge t(n) = a \qquad si\ p = a$$

$$n \in D_t \wedge t(n) = a \wedge (\forall i : 1 \leq i \leq n : Match(p_i, t, n.i) \qquad si\ p = a(p_1, \ldots\ldots, p_n)$$

Exemple 2 : voir ci-dessous: soit l'arbre objet t = a(b(c), a(b(c), a(c, c))) et le motif d'arbre p = a(b(c), υ) ci-dessous , la fonction Match(p, t, n) est vérifiée uniquement pour n = ε et n = 2. Match (p, t, ε) est vérifiée car t/ε = a(b(c), a(b(c), a(c, c))) et v = a(b(c), a(c, c)) . La même chose pour Match (p, t, 2), t/2 = a(b(c), a(c, c)) et v = a(c, c).

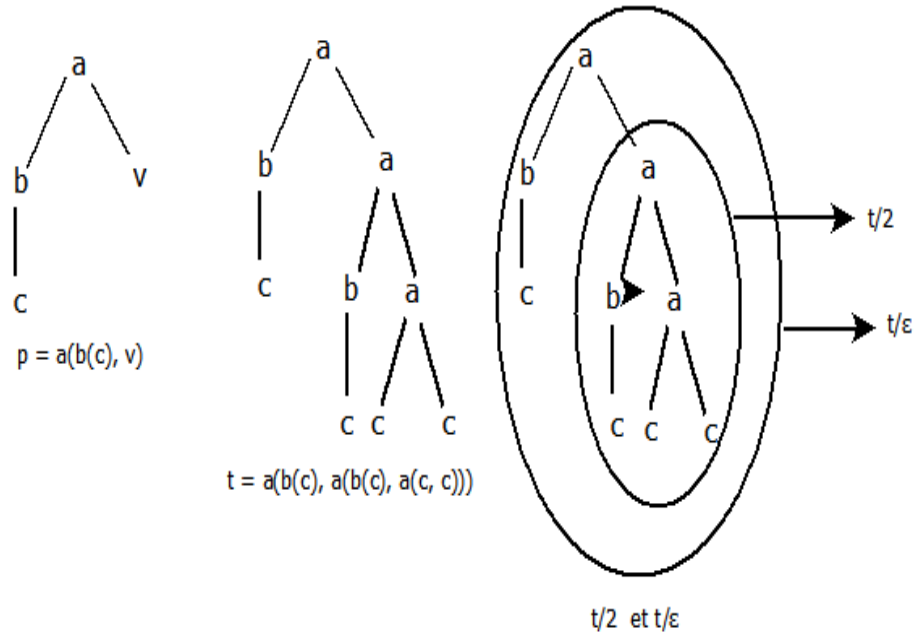


Figure II.6: le principe de la fonction $Match(p, t, n)$ pour extraire le motif p

4. Les algorithmes de recherche de motifs d'arbres (RMA)

Il y a plusieurs algorithmes de RM. La différence entre ces algorithmes est au niveau de représentation de l'arbre motif et de l'arbre objet. Cleophas(l'auteur de [12]) construit une taxonomie des différents RMA, il a divisé ces algorithmes en trois catégories, les algorithmes utilisant les automates d'arbre,les algorithmes basés sur la notion de 'match set' et les algorithmes basés sur la notion de 'stringpath matching'.la division de Cleophas est basée sur la méthode de travail de l'algorithme mais en pratique il y a deux .soit vous utilisez l'arbre(recherche dans l'arbre) soit vous utilisez un texte qui caractérise l'arbre (recherche dans un texte).vous trouver dans cette section l'algorithme naïf de recherche de motif , L'algorithme de Knuth-Morris-Pratt , L'algorithme d' Aho-Corasick(AC), l'algorithme descendant de Hoffmann et O'Donnell (1982), Ramesh et Ramakrishnan (1992).

4.1. L'algorithme naïf de recherche de motif :

L'algorithme de RMA naïf c'est l'algorithme de base de la RMA, il ressemble à l'algorithme de recherche de motifs dans un texte(RMM) mais il fonctionne de la manière suivante :Pour chaque nœud de L'arbre objet, nous testons si ce nœud est une occurrence du motif ou non ? Ce Test est effectué en comparant les nœuds du motif avec les nœuds de l'arbre Objet de gauche à droite. Si tous les nœuds du motif sont égaux aux nœuds de l'arbre objet, une occurrence a été trouvée et la position de cette occurrence est retournée. Sinon, la recherche se poursuit en passant au nœud suivant de l'arbre objet. La complexité de l'algorithme naïf est de l'ordre de $O(n.m)$ ou n est la taille de l'arbre objet et m est la taille du motif ,lorsque le motif et l'arbre objet ont les mêmes tailles la complexité égale $O(n.n)$.

4.2. L'algorithme de Knuth-Morris-Pratt

L'algorithme Knuth-Morris-Pratt [13] a été fait pour la RMM il est très utilisé pour l'extraction de connaissance à partir de texte, il sert à calculer l'occurrence et le positionnent de motif de mots p (modèle de chaine) dans un texte objet t.Si on considère que $T[i]$ donne le caractère de la position i dans le texte objet t et $p[j]$ donne le caractère de la position j dans le motif p.L'algorithme naïf de RMM est comme suit voir ci-dessous. Lorsqu'on applique l'algorithme naïf on trouve qu'il y a toujours un problème de retour en arrière dans

le texte (chaque caractère est lu plusieurs fois) parce que il y a toujours un glissement complet de motif ($j = 1$) a pour chaque $T[i + j] \neq p[j]$.

Algorithme naïf

Pour ($i = 1$ a n **) faire**

Tant que ($T[i + j] == p[j]$ **et** $j \leq n$ **)**

faire

$j = j + 1$

fin Tant que

Si ($j > n$ **) alors**

Sortie (occurrence de P à la position (*i*)**)**

fin si

$j = 1$

fin pour

Algorithme Knuth-Morris-Pratt

Pour ($i = 1$ a n **) faire**

Tant que ($j \geq 0$ **et** $P[j + 1] \neq T[i]$ **)**

faire

$j = f(j)$

fin Tant que

$j = j + 1$

si ($j == m$ **) alors**

sortie (occurrence de P à la position (*i-m*))

$j = f(j)$

fin si

fin Pour

L'algorithme Knuth-Morris-Pratt (voir ci-dessus) sert à optimiser l'algorithme naïf, par un prétraitement exécuté au niveau de motif p. Pour construire un tableau $f[j]$(la fonction d'échec), indiquant pour chaque position j dans le motif p la prochaine position $f[j]$ dans le motif et non dans le texte, avec le tableau $f[j]$ le retour en arrière dans le texte pour vérifier les caractères qui ont été précédemment vérifiés n'existe pas (lorsque $T[i + j] \neq p[j]$),car il est remplacé par un retour dans motif p (un glissement dans le motif lui-même).voir ci-dessous l'algorithme de la fonction d'échec (le calcule de tableau $f[j]$) et un exemple que donne le tableau $f[j]$ du motif $p = \{ababaca\}$.

L'algorithme de la fonction d'échec

$f[0] = -1$

$f[1] = 0$

$k = 0$

Pour ($j = 2\ a\ m$**) faire**

Tant que ($k \geq 0$ **et** $p[k+1] \neq p[j]$**)**

faire

$k = f(k)$

fin Tant que

$k = k + 1$

$f(j) = k$

fin Pour

Exemple :

p =	a	b	a	b	a	c	a
j =	0	1	2	3	4	5	6	7
$f(j)$ =	−1	0	0	1	2	3	0	1

Concernant la complexité de l'algorithme Knuth-Morris-Pratt, il est à mentionner que, le prétraitement est égale à O(m) et l'algorithme de recherche est égale à O(n)

4.3. L'algorithme d'Aho-Corasick(AC)

L'algorithme d'Aho-Corasick [14] est utilisé pour la recherche de plusieurs motifs en une seule fois, en garantissant que chaque caractère de texte est lu une seule fois. Le prétraitement sur les motifs de l'algorithme AC permet de produire l'automate AC, qui est composé de l'arbre digital et les états finaux des mots, plus la fonction d'échec. L'arbre digital contient des nœuds représentant les états et des arêtes indiquant les caractères de motif. le passage d'un état à l'autre sera par la fonction goto($g(q_i, a) = q_j$ ou$q_i et q_j$sont des états, $a \in \Sigma$).Dans chaque nœud de l'arbre digital v on trouve $f(v)$la fonction d'échec et la fonction $out(v)$ qui sera égale { P_i} si v c'est d'état final pour la chaine de motif P_i .la construction de l'automate AC est exécutée en deux phases :

La première phase est divisée en deux

1. la construction de l'arbre digitale tel que $p = \{P_1, \dots, P_n\}$. on charge les motifs $P_1, \dots, P_n$ une par une. lorsque la suite des caractères de motif P_i ne trouve pas sa place dans l'arbre digital on crée une nouvelle branche dans l'arbre digitale. Il faut dans chaque nœud qui représente un état final pour un motif $P_i$de faire $out(v) = \{P_i\}$,v c'est le nœud actuel.
2. Compléter l'arbre digitale par l'ajout de la fonction de goto pour la racine c'est-dire dans quel cas je boucle sur l'état 0($g(0, a) \coloneqq 0$).

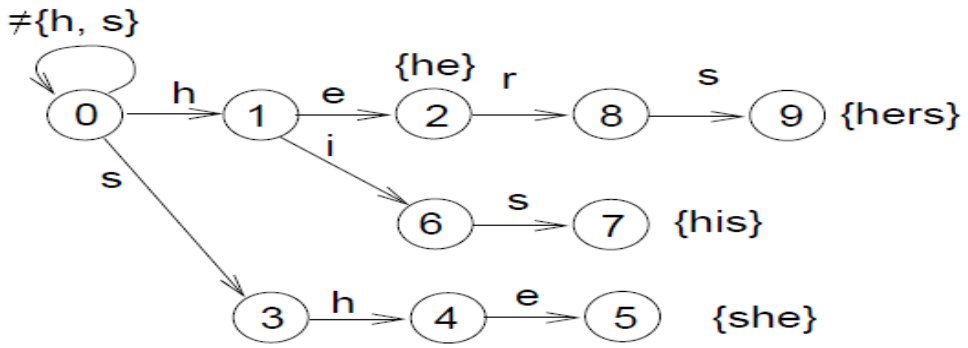


Figure II.7 :l'arbre digital la 1ère phase de motif $p = \{\mathbf{he}, \mathbf{she}, \mathbf{his}, \mathbf{hers}\}$

L'arbre produit par la première phase ne permet pas la recherche d'un ensemble de motifs, il reste de charger la fonction d'échec pour éviter de faire le retour en arrière dans le texte par exemple(voir l'arbre digitale de phase pour 1e motif $p = \{\text{he}, \text{she}, \text{his}, \text{hers}\}$) si g(4,e) est vérifiée par cette arbre, il faut marquer la fin de chaine « she » et faire le retour dans le texte pour récupérer la position perdue et détecter la fin de chaine « he ».Par la fonction d'échec ce problème est évité par une liaison(flèche) de passage de l'état 5 vers 2, donc on marque la fin de chaine « she » et on passe directement à l'état 2 pour marquer la fin de chaine « he ».

La deuxième phase de prétraitement de motif sert à faire le calcul de la fonction d'échec pour chaque état et par rapport aux autres états de l'arbre digital.

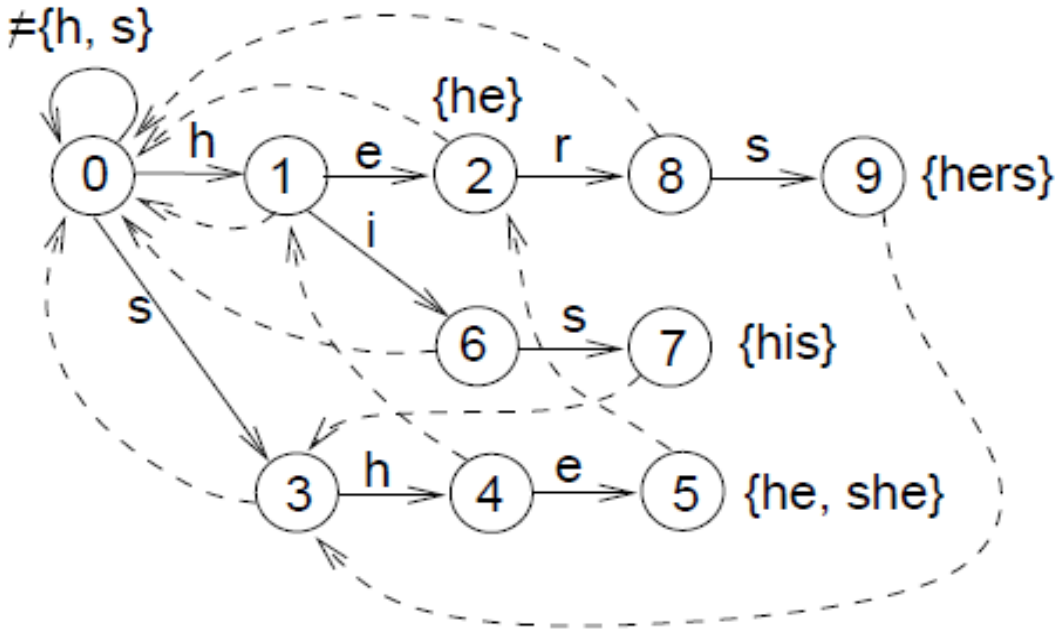


Figure II.8 : L'automate de AC de la 2ème phase de motif p= {he,she,his,hers}

L'algorithme de calcul de la fonction d'échec

Q := fils_vide();

Pour chaque ($a \in \Sigma$) **faire**

Si $(g(0,a) \neq 0)$ **alors**

$q = g(0,a)$

$f(q) := 0$

$enfiler(q, Q)$

finsi

fin Pour

Tant que $\boldsymbol{not}(\, fils\_vide(Q))$ **faire**

$r := D\acute{e}filer(Q)$

Pour chaque ($a \in \Sigma$) **faire**

Si $(g(r,a) \neq 0\,)$ **alors**

$u = g(r, a)$

$Enfiler\ (u, Q)$

$v := f(r);$

finsi

fin Pour

Tant que $g(v, a) = 0$ **faire**

$v := f(v); // (*)$

$f(u) := g(v;\ a);$

$out(u) := out(u) \cup out(f(u));$

fin Tant que

La RM par l'automate AC est effectuée à l'aide de ce pseudo code voir ci-dessous

$q := 0$

Pour $(i = 1\ a\ n)$ **faire**

Tant que $g(q, t[i]) = \emptyset$ **faire**

$q := f(q)$

fin Tant que

Si $(out(q) \neq \emptyset)$ **alors**

Ecrire(i,out(q))

finsi

fin Pour

Concernant la complexité de l'algorithme AC, il y a le prétraitement égale O(m) et l'algorithme de recherche dans l'automate qui comporte n comparaisons plus les comparaisons des motifs qui existent O($k\ n$) ,k le nombre d'occurrence de motif .

4.4. L'algorithme descendant de Hoffmann et O'Donnell (1982)

L'algorithme descendant de Hoffmann et O'Donnell [15] permet la RMA par un prétraitement exécuté sur le motif avant de commencer la recherche. L'idée de cet algorithme prétend considérer chaque chemin de la racine vers les feuilles de motif p comme une chaine de caractères dans laquelle les symboles de l'alphabet sont intercalés avec des nombres(les arités) indiquant quelle branche du père a été suivie.

Par exemple : le motif $p = a(a(b, v), c)$ admet l'ensemble des chaines de caractères suivants $\{a1a1b, a1a2, a2c\}$. La variable v n'est pas représentée dans l'ensemble des chaines de caractères, parce qu'il représente une variante dans l'ensemble d'alphabet.

Lorsqu'on génère tous les chemins (les chaines de caractères) depuis la racine vers les feuilles de motif p ,on fait appel à l'algorithme d'Aho et Corasick (l'algorithme précédant) pour produire un automate reconnaissant toutes les instances des chemins dans un arbre objet t.

Après ce prétraitement on fait la recherche d'après la méthode suivante : on fixe un nœud de départ dans l'arbre objet t(en début c'est le nœud racine), si il y a une correspondance entre le nœud de l'arbre objet t et le nœud de motif p (dans l'automate $g(0, T[i]) = q$) alors on passe à l'état suivant dans l'automate et on passe au nœud $\text{suivant}(\text{n}, \text{r}(\text{n}))$ dans l'arbre objet t ,ou n représente le nœud actuel et r(n) représente l'arite qui doit être suivi dans l'arbre objet t ,on obtient r(n) à partir de l'automate AC en choisissent l'arité la plus petite non visitée .Automatiquement s'il n'y a pas de correspondance dans le trie d'arité (le domaine de l'arbre motif et arbre objet) ou dans les caractères de motif et les caractères de l'arbre objet ,le nœud fixé en début ne représente pas une occurrence de motif p .L'opération de recherche de l'algorithme de descendant de Hoffmann et O'Donnell est comme l'opération de recherche de l'algorithme précédant, la seul différence c'est les arités qui sont ajouter pour dirigent les comparaisons .

4.5. L'algorithme de Ramesh et Ramakrishnan (1992)

L'algorithme de Ramesh et Ramakrishnan est utilisé aussi pour la RMA, mais il ne fonctionne pas avec le principe d'automate d'arbre. L'idée de l'algorithme est de transformer l'arbre de

motif et l'arbre objet en des séquences de caractères, caractérisant la forme de motif et la forme de l'arbre objet (on garde le trie de domaine d'arbres) .Avec cette méthode la séquence de l'arbre motif représente un mot, la séquence de l'arbre objet t représente un texte et le problème recherche de motif d'arbre sera un problème de RMM.

La séquence utilisée pour caractériser la forme des arbres c'est la chaine d'Euler, Une chaine d'Euler d'un arbre est obtenue par le marquage en aller et en retour de parcours préfixé de l'arbre, c'est-à-dire pour chaque arête entre deux nœuds (a, b) il y a une arête (b, a). Le mot d'Euler d'un arbre c'est la chaine d'Euler mais à la place des nœuds on trouve les étiquètes de ces nœuds. Par exemple soit le motif $p = a\big(b(v_1, v_2), c(v_3, k)\big)$, voir ci-dessous dans la figure( ) comment obtenir la chaine d'Euler C_p et le mot d'Euler (noté par E_p)

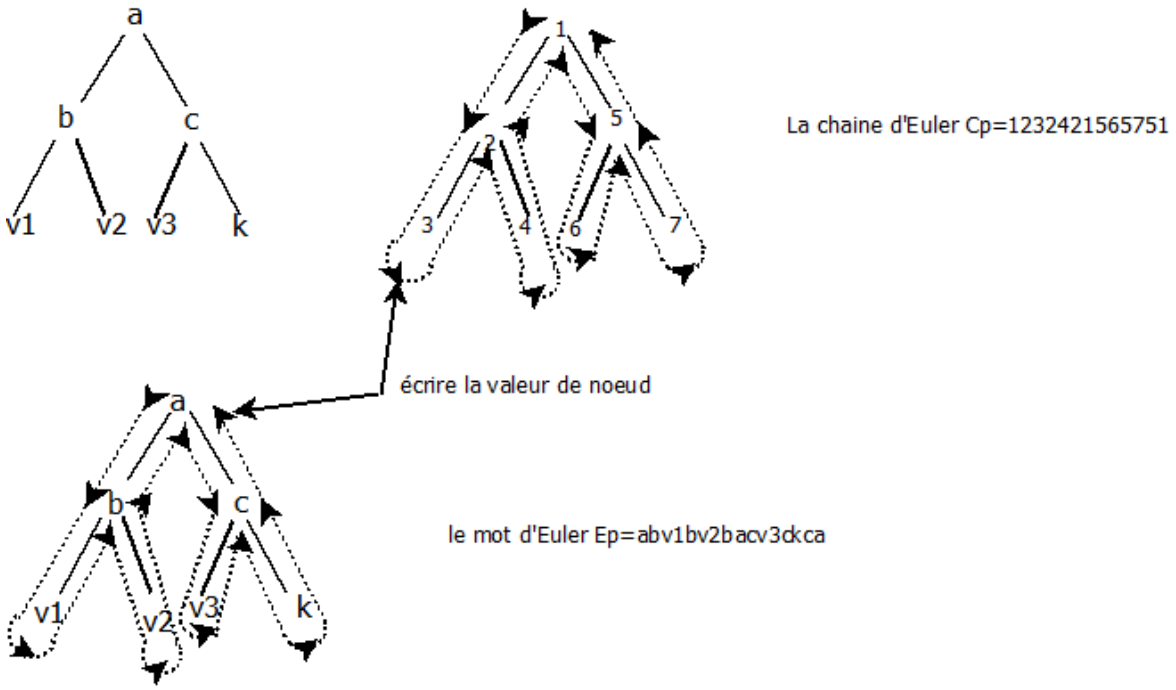


Figure II.9 : la chaine d'Euler d'un arbre

Dans les chaines d'Euler on trouve les propriétés suivantes :

- Les feuilles d'un arbre apparaissent une seule fois dans la chaine
- Une sous-chaine d'Euler commence de la première occurrence vers la dernière occurrence d'un nœud
- La chaine d'Euler d'un nœud « *a* » qui admet *k* fils,admet $k + 1$ fois le caractère « a »

L'algorithme est exécuté en deux phases, la 1ère phase c'est pour le prétraitement qui sert à faire la linéarisation des arbres (l'arbre motif et l'arbre objet). La 2ème phase c'est pour la recherche de motif.

- La 1ère phase :

1. Transformer l'arbre motif p pour obtenir le mot E_p , remplacer chaque variable $v_1, \dots, v_k$ Dans le mot E_p par un espace vide (pour l'exemple précédant E_p = « ab b bac ckc »)
2. On fractionne E_p en $k+1$ mot dénoté par $\sigma_1, \sigma_2, \dots \sigma_{k+1}$ (pour exemple précédant σ_1 =ab , $\sigma_2 = b$, $\sigma_3 = ba$, $\sigma_4 = ckca$)
3. Charger les chaines $\sigma_1, \sigma_2, \dots \sigma_{k+1}$ dans les tableaux booléens $M_1, M_2, \dots M_{k+1}$, la taille de chaque tableau M_i =$|E_s|$,(pour l'exemple précédant le résultat est

$M_1 = \frac{ab}{1100000000000}$, $M_2 = \frac{b}{0001000000000}$, $M_1 = \frac{bac}{0000011100000}$, $M_1 = \frac{ckca}{0000000001111}$)

A la fin de la 1ère phase on remarque que σ_1 représente un préfixe de E_p et la recherche est effectuée dans le tableau M_1 et dans les entrées non nulles de M_1 qui correspondent à la première occurrence d'un nœud. Il convient de dire si l'arbre de motif p existe dans l'arbre objet t, alors la séquence $\sigma_1 v_1 \sigma_2 v_2 \dots \sigma_k v_k \sigma_{k+1} v_{k+1}$ existe dans E_s et v_1 , v_2 , ..., v_{k+1} Sont des sous-mots de E_s , qui commencent par la position $j + |\sigma_i| + 1$ dans E_s. Avec j la positon actuelle dans E_s et $|\sigma_i|$ c'est la longueur de σ_i .

- La 2ème phase :

C'est la phase de recherche qui se déroule de la manière suivante :

L'existence de mot σ_i dans E_s signale la recherche de mot σ_{i+1} , c'est-à-dire on vérifie que les caractères de σ_i qui admettent les positions vraies (la valeur 1) dans le table M_i ,existe dans E_s .si la vérification est vraie on passe à la vérification des caractères de σ_{i+1} dans la table M_{i+1} à partir de la position $h + |\sigma_i| + |v_i| + 1$, avec $|v_i|$ la longueur de sous-mot v_i , h la position de la dernière commence de la recherche de σ_i dans E_s. Si la vérification est fausse (σ_i n'existe pas dans E_s) automatiquement il n'y a pas de motif). A l'arrivée de la fin de la table M_{k+1} on marque une occurrence de motif)

Concernant la complexité, il faut que k+1 tables de tailles $|E_s|$ doivent être construites. La complexité égale $O(n.k)$ on ajoute le prétraitement de l'algorithme qui est de l'ordre $O(m)$ ou m c'est le nombre de nœud de motif et de nœud de l'arbre objet.

5. Analyse et synthèse

D'après l'étude des algorithmes RM et les différents concepts et définitions de la théorie de la l'arbre, on remarque les points suivants :

1. La définition (1) d'arbre et de domaine d'arbre est plus important car Pratiquement (physiquement en mémoire) le domaine représente l'adresse mémoire de nœud et l'arité représente le fils suivant pour cette adresse, Pour plus détail voir la figure II.3 ci-dessous qui explique le fond pratique de cette définition.

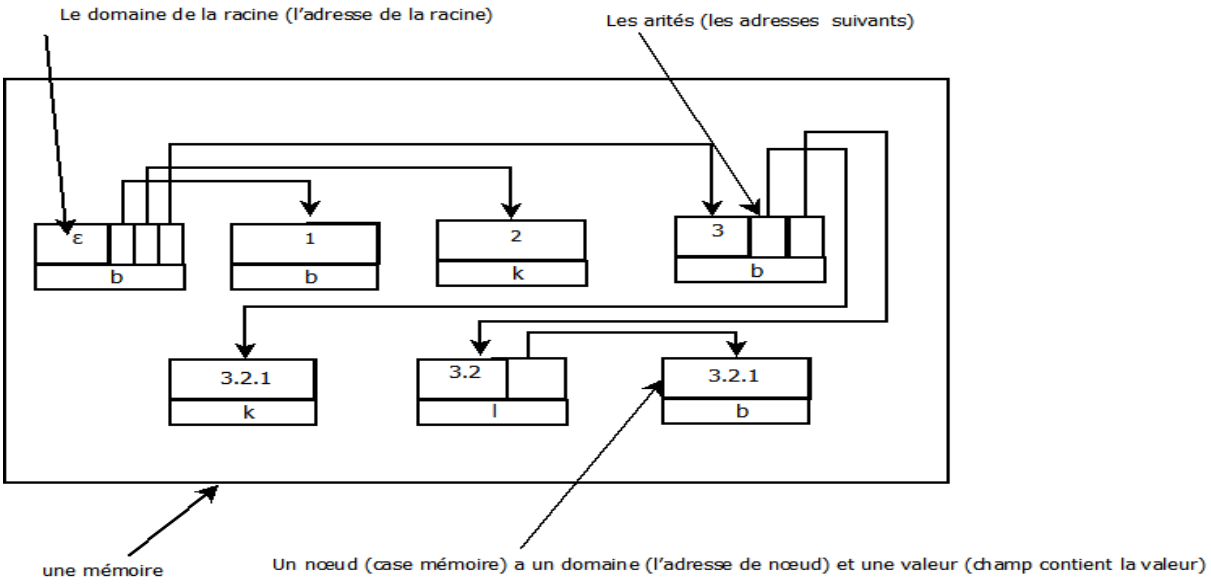


Figure II.10 : le Domain d'arbre en mémoire

2. l'opération de la RMA comporte deux sous-opérations, la 1ère est une recherche exacte qui signale le début de la 2ème, la 2ème c'est pour la recherche de l'existence d'un domaine (recherche de variable). Par l'observation on remarque que la 2ème opération est incluse dans la 1er. pour éclaircir. Soit par exemple le motif $p = a(b, c(v1, v2)$, la 1er opération c'est la recherche exacte de motif $p = a(b, c)$, ensuite si la partie exacte est trouvée on passe à la 2ème c'est la recherche des

domaines $v1$et $v2$ qui sont (2.1,2.2).Remarquez ici, si je connais à l'avance les valeurs (les étiquètes) possibles pour les domaines des nœuds $v1$et $v2$ la recherche sera une recherche exacte. Dans mon exemple supposons que $v1$et $v2$ sont $v1 = \{a, b\}$ $v2 = \{k, l\}$ la recherche sera comme suite, voir dans la figure II.11 ci-dessous.

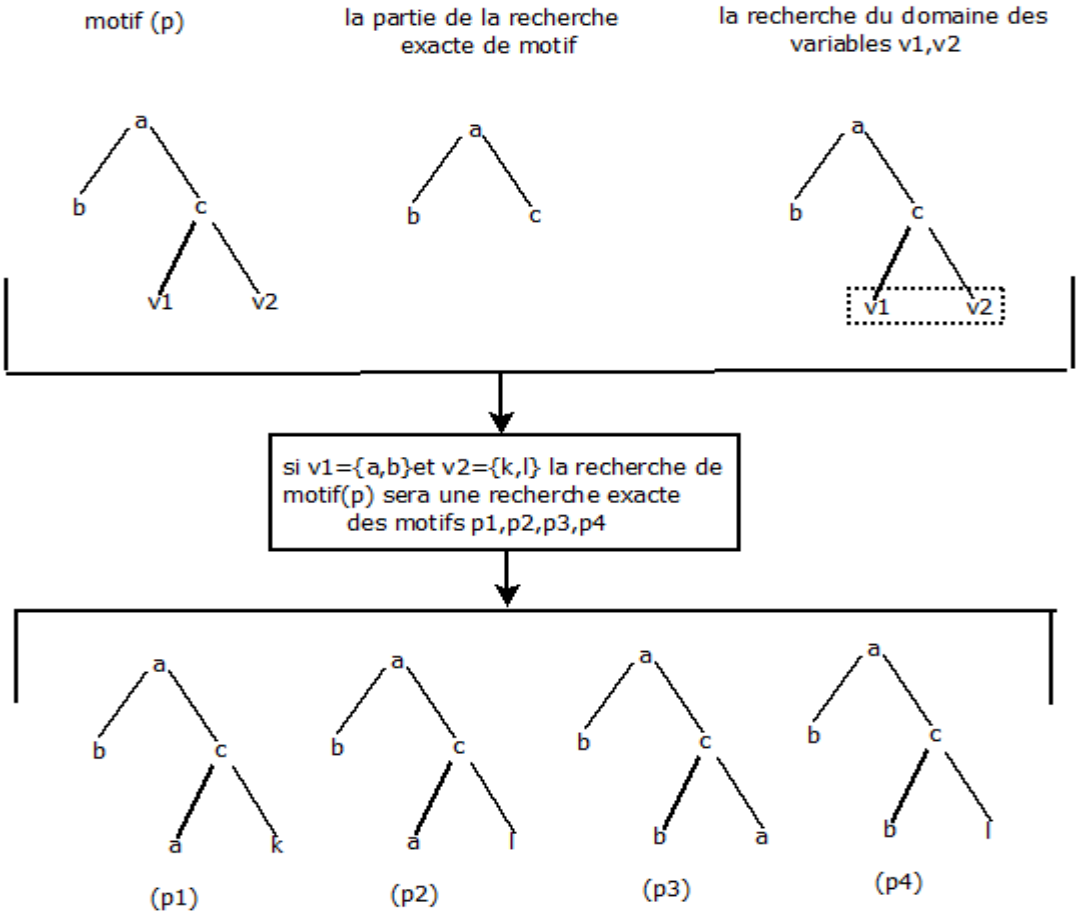


Figure II.11 : le motif exacte et le motif variable

3. On remarque une liaison étroite entre les algorithmes de RMA et les algorithmes de RMM .A partir de l'utilisation des algorithmes de RMM dans les algorithmes de RMA. Lorsqu'on cherche à appliquer un RMA on se trouve toujours devant deux possibilités soit l'utilisation d'une structure hiérarchique (l'arbre lui-même) soit l'utilisation d'une structure séquentielle qui représente un parcours de l'arbre (texte caractérisant l'arbre)
4. on vu que l'intersection des algorithmes de recherche de motif c'est le parcours de domaine de recherche (lors de la recherche il faut traverser tout le texte t ou tout l'arbre objet t).

5. La linéarisation (la chaine d'Euler) de motif d'arbre rend le motif d'arbre un ensemble de motifs de mots

Conclusion

Dans ce chapitre étude vient de définir et identifier, la structure d'arbre, les points théoriques essentiels pour la RMA, le problème de RMA et les algorithmes de RM. Pour la résolution de problème de RMXML, on remarque que l'algorithme naïf et le plus proche pour l'application dans le problème car La forme arborescente de XML à plus d'acceptabilité pour RMA, ainsi que la résolution de ce problème avec ce dernier assure l'utilisation des autres algorithmes de RMA. Le chapitre suivant sera une partie pour faire la relation entre, les algorithmes de RMA qui sont présentes par l'algorithme naïf et les outils XML défini dans le chapitre 1 dans but de résoudre le problème de RMXML.

Chapitre III : Etude expérimentale

Introduction

L'implémentation de la RMXML est en relation avec le fond théorique RM et le fond pratique et technique de XML, il est évident qu'il faut établir un lien entre ces deux. L'idée c'est de décomposer les théories (algorithme naïf) et les techniques (recherche avec Xpath) en unités pour voir les points forts et les points faibles. Lorsque ces points sont clairs, il faut faire l'assemblage des unités par rapport au problème et par l'augmentation des points forts. Ce chapitre n'est que le résultat de cette technique.

Ce chapitre comporte, les outils et les environnements utilisés, l'architecture de l'application réalisée pour la recherche de motif XML et pour les preuves de la méthode utilisée, une partie d'analyse sur la recherche de motif dans le contexte XML, une partie de teste et discussion sur les travaux pratiques réalisés, et la synthèse de la méthode utilisée dans un futur prochain.

1. Les outils et environnements utilisés

1.1. Oxygen

« <oXygen/> XML Editor est une application multi-plateforme disponible sur tous les principaux systèmes d'exploitation (Windows, Mac OS X, Linux, Solaris) et peut être utilisé de manière autonome ou comme un plugin Eclipse » [16], <oXygen/> est un logiciel shareware, on utilise ce outil pour faire la lecture des documents XML et pour l'analyse et la vérification des solution des requête Xpath. L'arbre de XML et le motif XML sont chargé avec cet outil pour assurer la syntaxe XML.

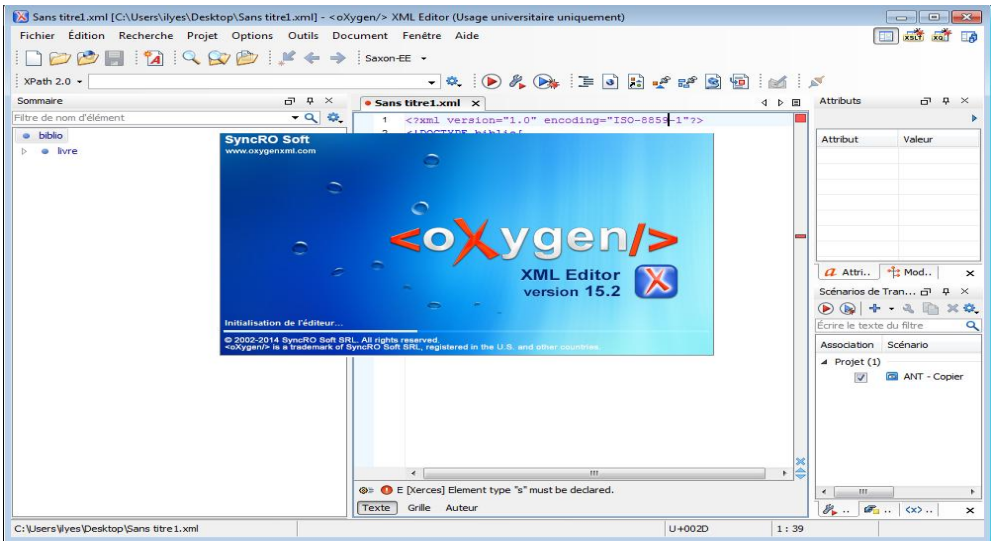


Figure III.1 : l'interface de l'outil <oXygen/>

1.2. Visuel Basic et DOM

« **Visual Basic (VB)** est un langage de programmation événementielle de troisième génération ainsi qu'un environnement de développement intégré, créé par Microsoft pour son modèle de programmation COM. Visual Basic est directement dérivé du BASIC et permet le développement rapide d'applications, la création d'interfaces utilisateur graphiques, l'accès aux bases de données en utilisant les technologies DAO, ADO et RDO, ainsi que la création de contrôles ou objets ActiveX.» [17]. On utilise le visuel basic pour faire la programmation de XML à l'aide de la bibliothèque MSXML (voir l'annexe) qui rendre la forme textuel de XML se forme d'un arbre en mémoire (arbre DOM). On remarque que la bibliothèque de l'arbre DOM dans les autres langages fonctionne avec le même principe avec MSXML, l'arbre DOM ou XMLDOM c'est un standard pour le traitement du XML en mémoire.

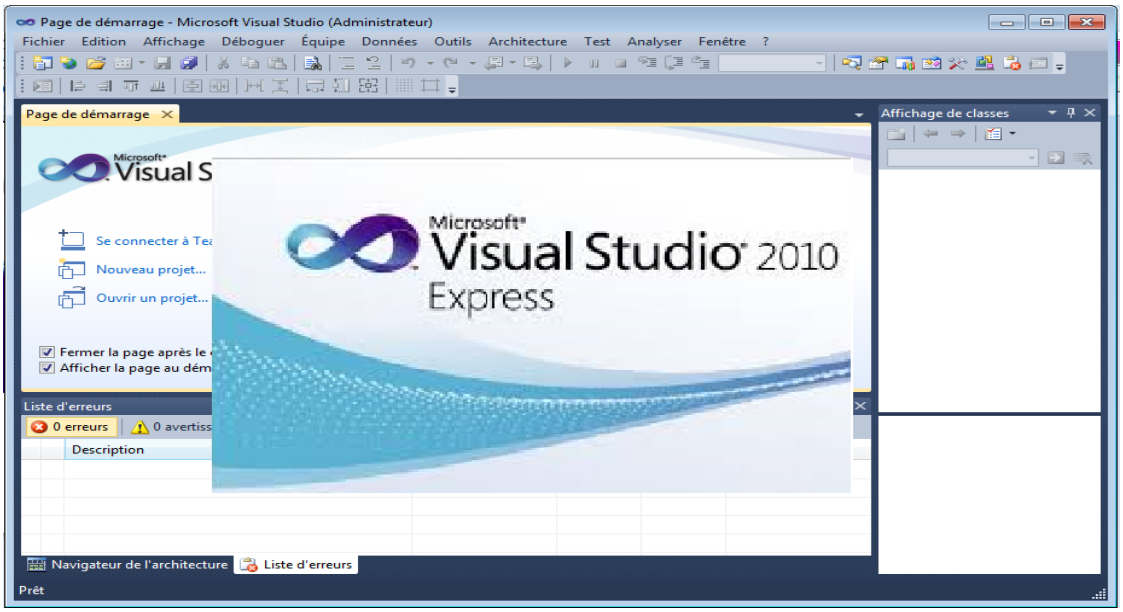


Figure III.2 : l'interface de l'environnement visuel studio express

Remarque : on a utilisé visuel studio 2010 express en tant que une version gratuite proposée et conçue pour l'étudiant. On remarque aussi que tous les offres du standard DOM c'est des fonctions de manipulation de l'arbre XML n'est la structure exacte (les pointeurs, les champs …etc), la plupart des outils XML (Xpath) sont développés avec ce standard de plus ils sont intégrer pour l'utilisation dans ce dernier.

2. L'architecture de l'application réalisée

On a réalise une application comportant les opérations utilise pour la partie d'analyse. L'application est exposée à l'aide d'une fenêtre **menu** qui permet l'accès à la fenêtre de gestion de XML (avec la Bouton prouve) et à la fenêtre de la recherche de motif (avec la Bouton la recherche de motif XML).

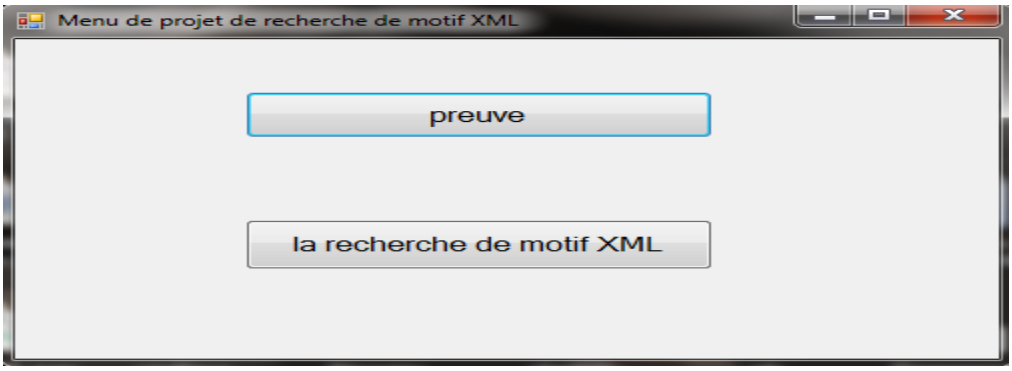


Figure III.3 : la fenêtre de menu de l'application réalisée

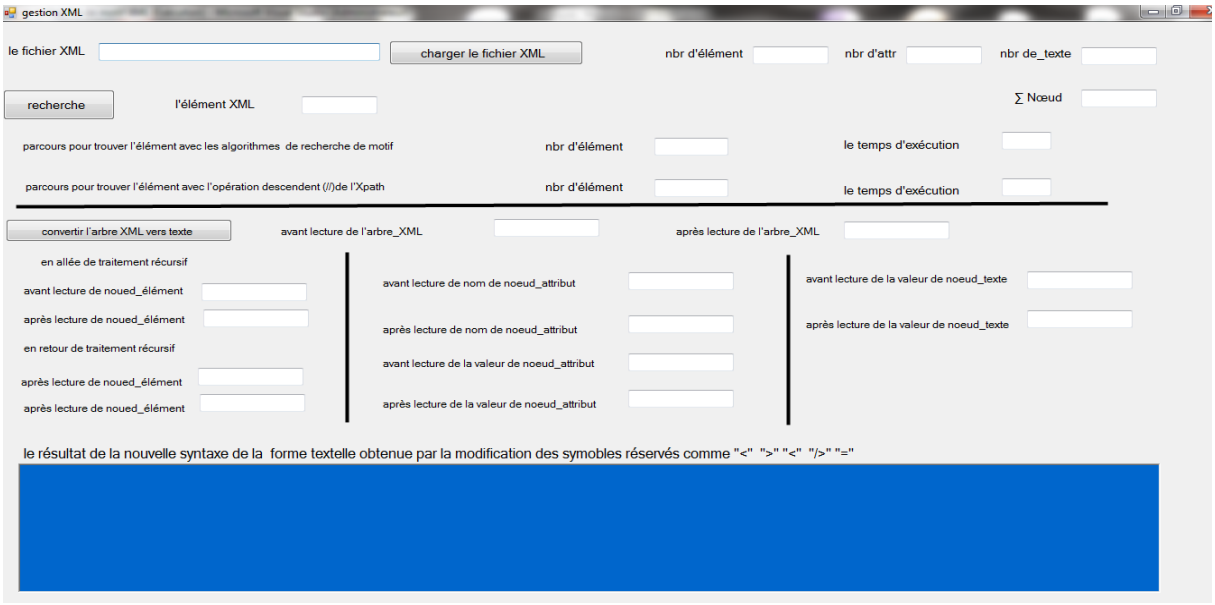


Figure III.4 : la fenêtre de la partie preuve de l'application réalisée

La fenêtre gestion XML permet de charger, les programme utilisé pour prouvé que l'Xpath est un meilleur outil pour la sélection des nœuds, les programme qui modifier la syntaxe XML et de prouve que la forme textuel de XML est obtenu par le parcours de l'arbre DOM de l'arbre XML ainsi le programme de calcul des nœuds de l'arbre XML (élément, texte, attribut)

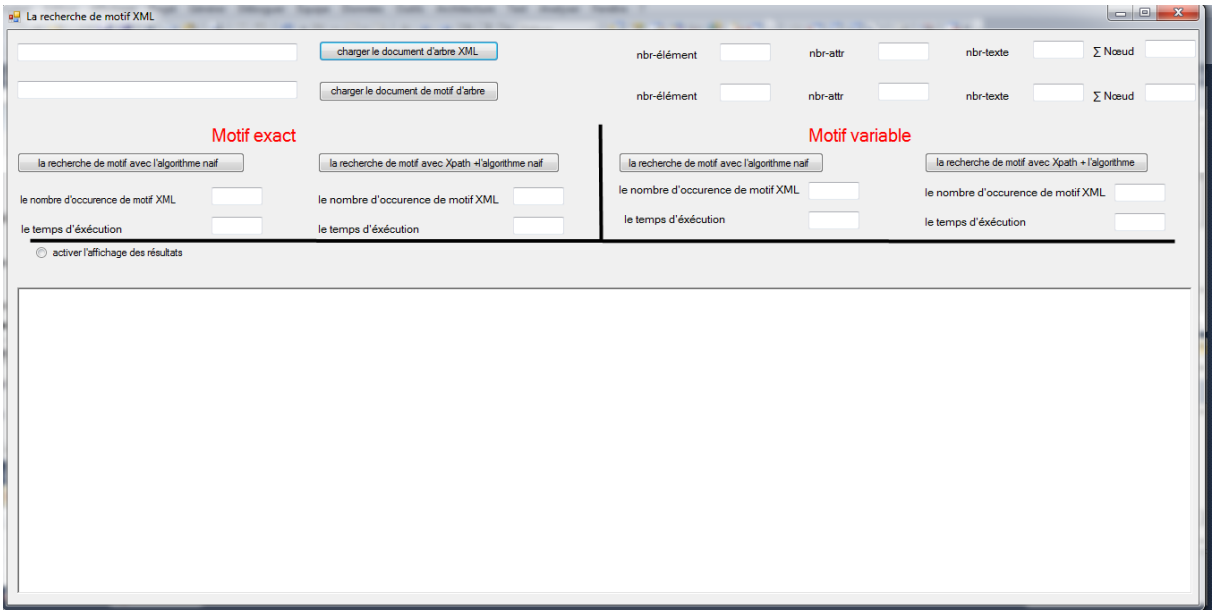


Figure III.5 : la fenêtre de la recherche de motif de l'application réalisé

La fenêtre de RMXML charge, les programmes de recherche de motif exacte et de motif variable avec l'algorithme naïf et les mêmes programmes mais avec Xpath+ l'algorithme naïf

3. Analyse sur la recherche de motif dans le contexte XML

3.1. La forme textuelle de XML et les outils XSD et DTD

La forme textuelle de XML c'est une forme obtenue à l'aide d'un parcours préfixé récursif de l'arbre DOM de XML et le marquage en aller et en retour des nœuds non feuille (texte et attribut). Ce parcours est à peu prés comme le parcours de la chaine d'Euler, voir la figure III.6 ci-dessous et la figure II.9 (de la chaine d'Euler)

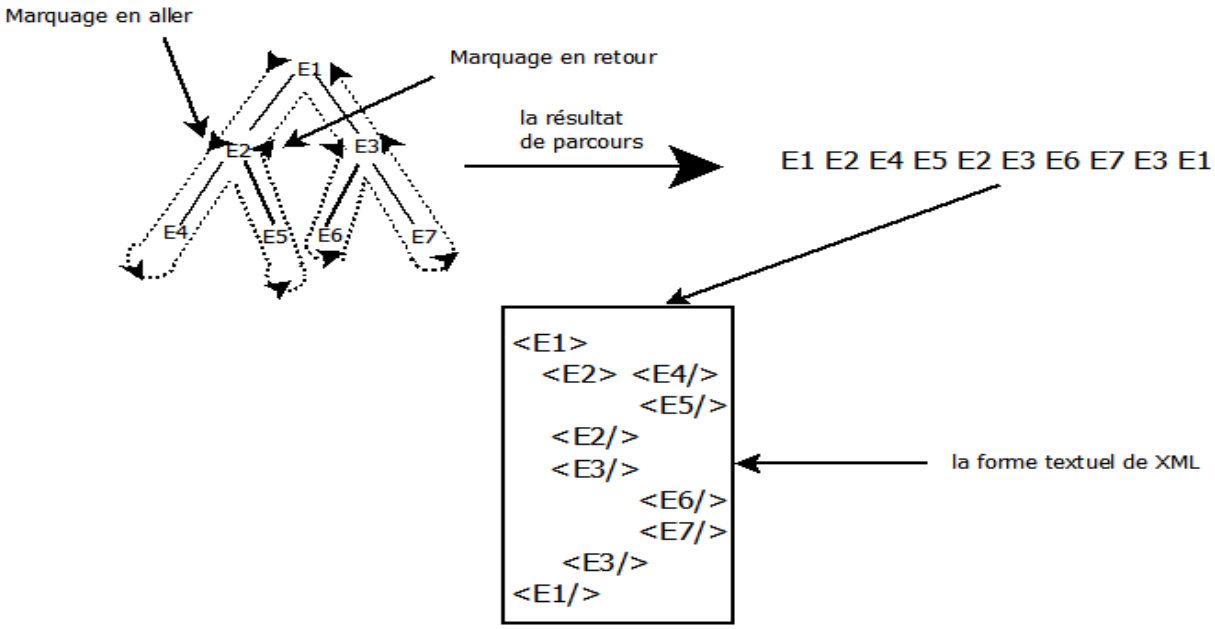


Figure III.6 : le parcours pour obtenir la forme textuelle de l'XML

On a essaye de réaliser une fonction permettant de fait ce parcours pour obtenir la structure XML avec des syntaxes différent (forme textuel avec un changement sur symbole réservé « < » « > » « /> » « = »).

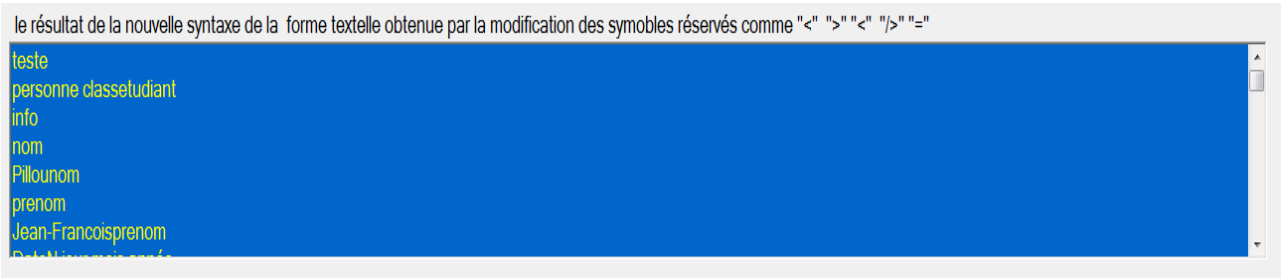


Figure III.7: le résultat de fonction conversion l'arbre XML vers texte

La fenêtre gestion XML (de la figure III.4) permet de modifier la syntaxe XML à l'aide de la Bouton « convertir l'arbre XML vers texte » et par le remplissage des champs pour les remplacer les symboles réserver de XML« < » « /> » « = »…etc., dans le ces ou le remplissage n'existe pas le résultat c'est l'écriture des valeurs des nœuds seulement. Voir la figure III.7

Les outils XSD et DTD servent à faire la définition d'un document XML par la représentation des règles de production de l'arbre XML, mais on peut utiliser ces outils pour calculer l'existence d'un chemin ou d'un nœud dans la structure de document XML. Par exemple (voir dans la figure suivant) un arbre XML et la DTD de ce arbre XML. On peut calculer les étiquètes des nœuds variables de motif à partir de DTD et avant de lancer la recherche dans l'arbre XML et avec seulement la reconnaissance du père de la variable dans le motif.

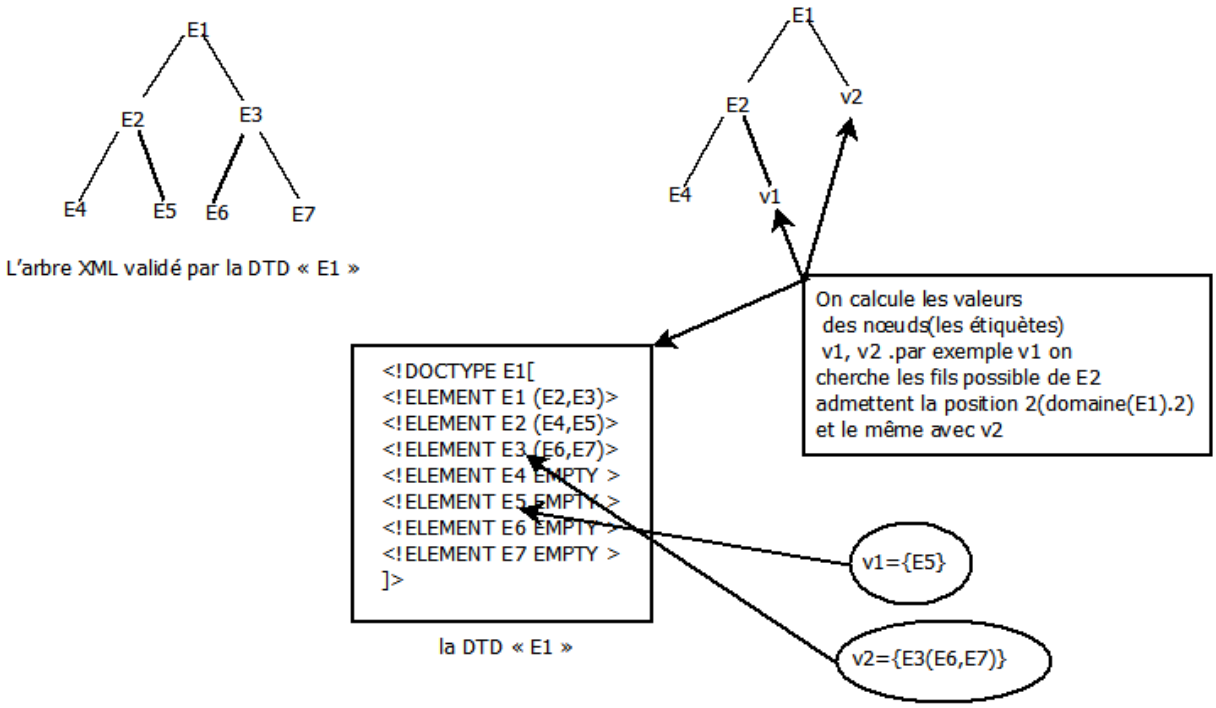


Figure III.8 : Comment trouver des nœuds variables d'un motif (avec DTD)

Remarque : L'application de cette méthode n'est pas simple car on trouve des variantes dans la DTD ou XSD (les indicateurs d'ordre, les indicateurs de répétitions…etc.)

D'après ces points, l'application des algorithmes de recherche de motif d'arbre pour la RMXML, demande la reconnaissance de type de nœud avant de lancer un traitement.

3.2. La sélection des nœuds avec Xpath et avec les algorithmes de RMA :

Dans cette section on suppose que la recherche de motif c'est une recherche d'un nœud dans l'arbre XML. Xpath dans cette hypothèse est fonctionne à l'aide de la fonction

descendant (// élément cherché), qui représente normalement la méthode de recherche des algorithmes de recherche de motif d'arbre. Parcours de l'arbre jusqu'a trouver l'élément cherché. On a réalisé La fonction de parcours (recherche élément) de l'arbre XML avec l'arbre DOM et l'autre fonction avec l'arbre DOM et par utilisation de Xpath par instruction selectnodes(chemin xpath) voir ci-dessous le teste et le résultat :

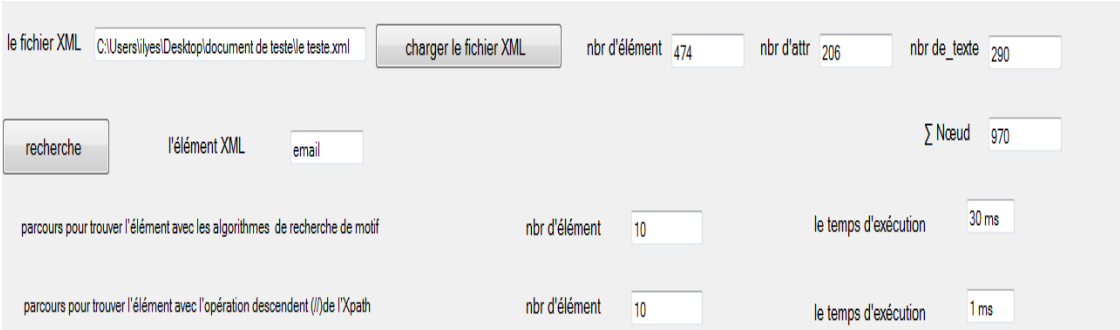


Figure III.9 : Le résultat de teste entre Xpath et le parcours des algorithmes de recherche de motif d'arbre

On a trouvé comme résultat dans un fichier de 970 nœud que Xpath sélectionne l'élément chercher en 1ms par contre l'algorithme de parcours qui est utilisé par les algorithmes de recherche de motif d'arbre en 30 ms voir la figure III.9 .

La justification : le principe de fonctionnement de Xpath en mémoire est inconnu, Xpath c'est un outil de base de la recommandation W3C est toute l'offre de cette dernière c'est l'interface qui permet de charger les requête et de rendre les résultats. Avec la petite expérience dans ce domaine on constate que Xpath utilise l'arbre DOM mais avec des champs d'adresse spéciale. Voir ci-dessus dans la figure notre justification.

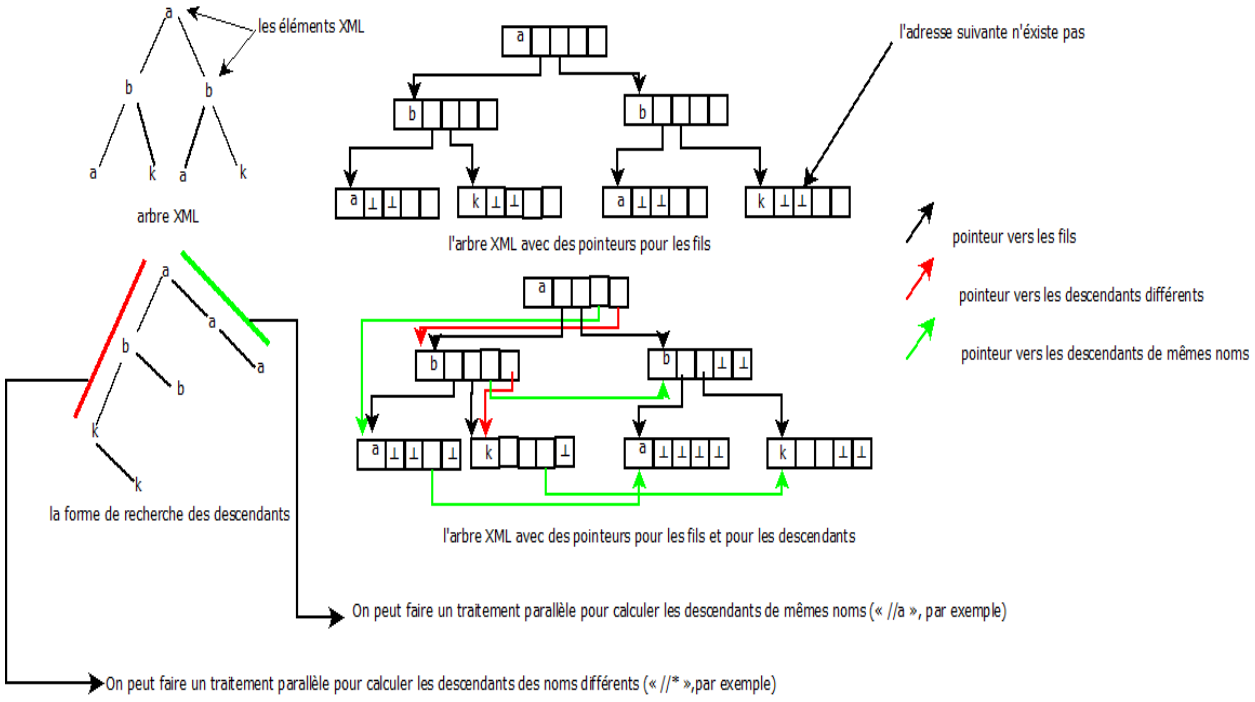


Figure III.10 : la structure de recherche de Xpath dans l'arbre DOM

Théoriquement avec cette structure la complexité de la recherche au pire cas égale $O(n)$et en moyen $O(\log n)$ on ajoute les traitements de l'algorithmique parallèle qui sont possibles avec cette structure et qui permettent de diminuer cette complexité par le croisement des processeurs.

Maintenant en replace l'hypothèse précédente par le problème réel de la recherche de motif XML.la solution avec Xpath c'est d'exprimer l'arbre motif par un chemin de recherche dans arbre XML en utilise les axes de Xpath fils« / », descendant « // », parent «..» plus les requêtes d'attribut et de texte. Voir ci-dessous dans la figure III.11 exemple comment faire.

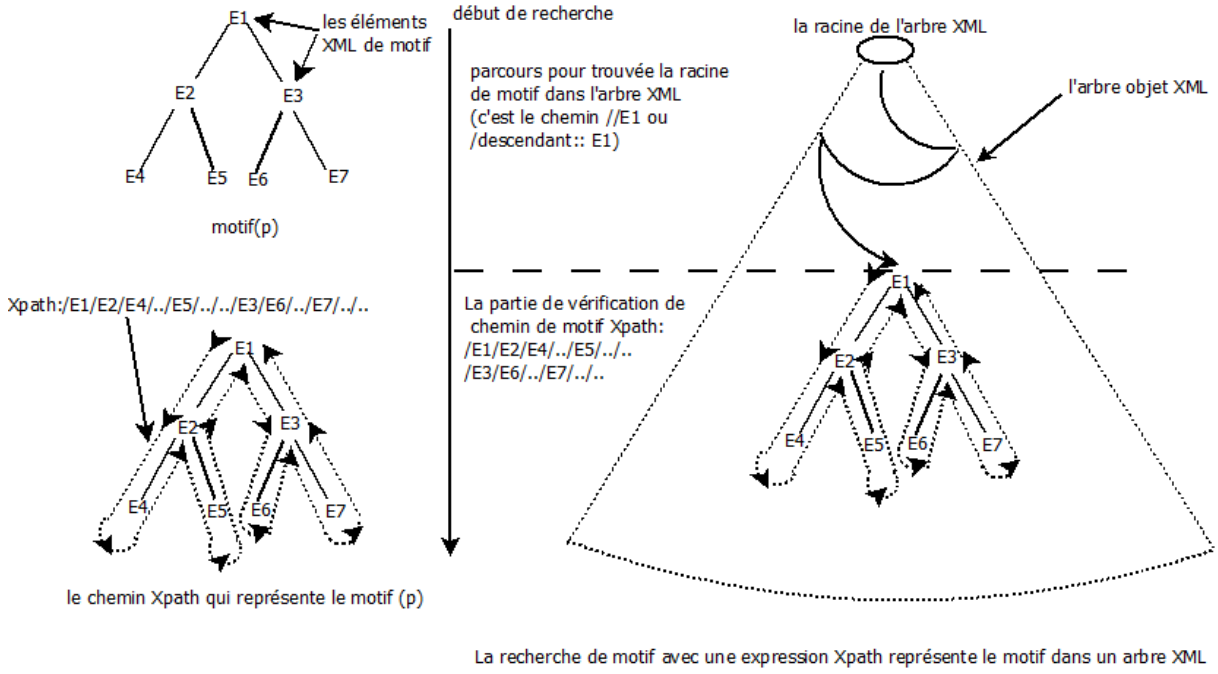


Figure III.11 : la recherche de motif XML avec l'outil Xpath

La convolution sur les chemins de motif de cette manière n'est pas possible pour chaque motif d'arbre,car Lorsque j'ai un motif admettent les même noms d'élément cette idée ne fonctionne pas par ce que il ne peut pas la différence entre deux élément frère de même nom voir la figure III.12 ci-dessous.

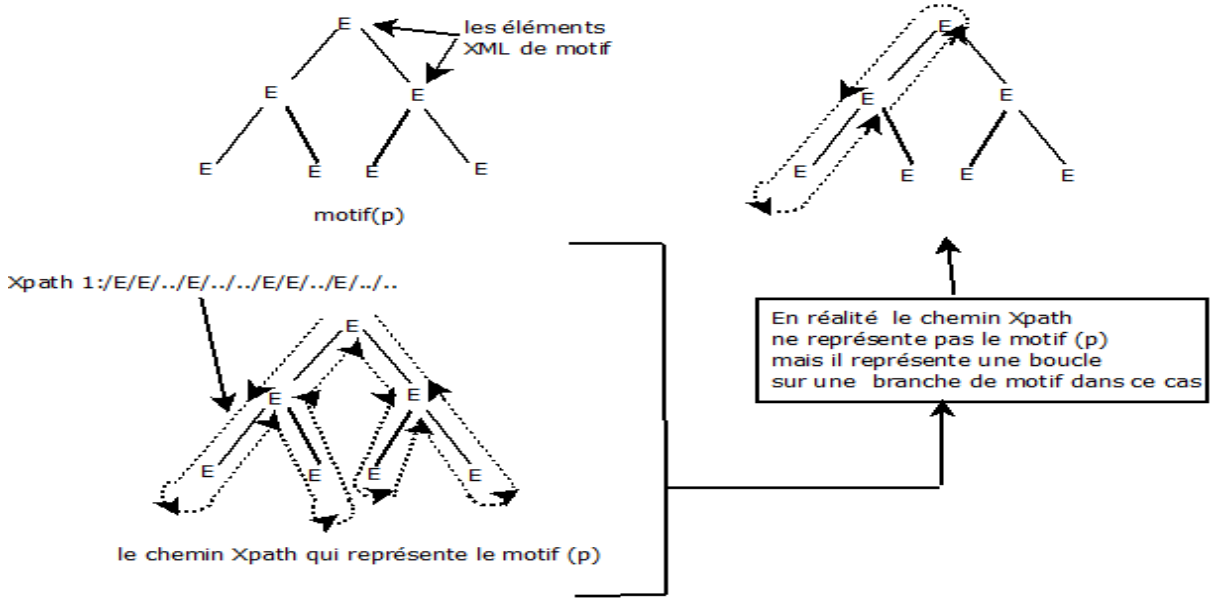


Figure III.12 : Le calcule avec l'Xpath seulement n'est pas possible

D'après ces points on conclure que Xpath est un meilleur outil pour la sélection des nœuds dans l'arbre XML et n'est pas pour les motifs (ensemble de nœuds).

3.3. La recherche de motif XML exacte et variable

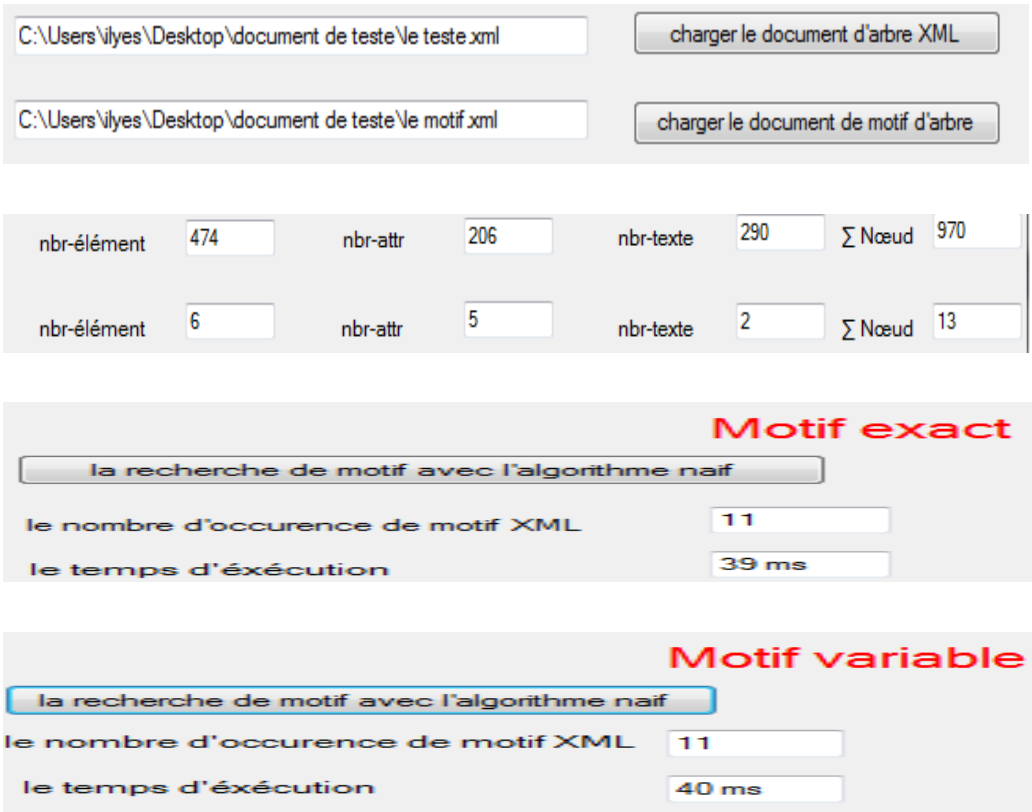
Il y a deux types de recherche de motif, la recherche de motif exacte : chercher l'existence d'un document XML (motif) dans un autre(arbre XML), la recherche variable : c'est la recherche de motif exact avec l'utilisation de variable pour indiquer la variance des nœud de motif .Il y a plusieurs modèles de motif variable de XML voir ci-dessous le tableau III.1 des différents modèles variables pour un nœud élément admet un texte et un attribut ainsi les mots réservées pour indiquer la variance des nœuds de la fonction de recherche de motif variable.

Les modèles des nœuds variables de XML	Explication
<v a="attribut exacte"> texte exacte </v>	Chaque nœud qui admet n'importe quel nom d'élément et qui admet un attribut a="attribut exacte" et un texte exacte
<v a="attribut exacte">tv</v>	Chaque nœud qui admet n'importe quel nom d'élément et qui admet un attribut a="attribut exacte" et un texte variable
<v a="attribut exacte">cv</v>	Chaque nœud qui admet n'importe quel nom d'élément et qui admet un attribut a="attribut exacte" et qui admet n'improte quel contenu « cv »
<v >cvav</v>	Chaque nœud qui admet n'importe quel nom élément et n'importe quel ensemble d'attribut et avec n'importe quel contenu « cvav »

<v a1="v"> texte exacte </v>	Chaque nœud qui admet n'importe quel nom d'élément et qui admet un attribut admet n'importe quel nom et n'importe quel valeur d'attribut et un texte exacte

Tableau III.1 : les mots réservés pour indiquer la variance de nœud élément, texte et attribut

La RMXML exact avec l'algorithme naïf est basée sur deux fonctions .La fonction « parcours » qui permet de parcourir l'arbre XML et la fonction vérifier motif qui permet pour chaque nœud fait une vérification de l'existence de motif. Le même principe pour La RMXML variable est aussi basé sur deux fonctions mais ils travaillent avec les mots réservés pour le motif variable. Voir les champs et les Bouton concernée par ces fonctions ainsi les résultats de RM.



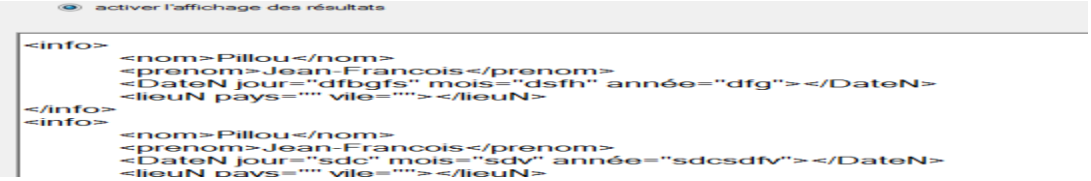


Figure III.13 : les différents champs et Bouton de la recherche avec l'algorithme naïf de motif exact et variable XML

3.4. L'amélioration des algorithmes de recherche de motif avec Xpath

L'amélioration des algorithmes RM est fait avec la fusion de Xpath en tant que un chercheur de la racine de motif voir ci-dessous la valeur a ajouter par cet outil

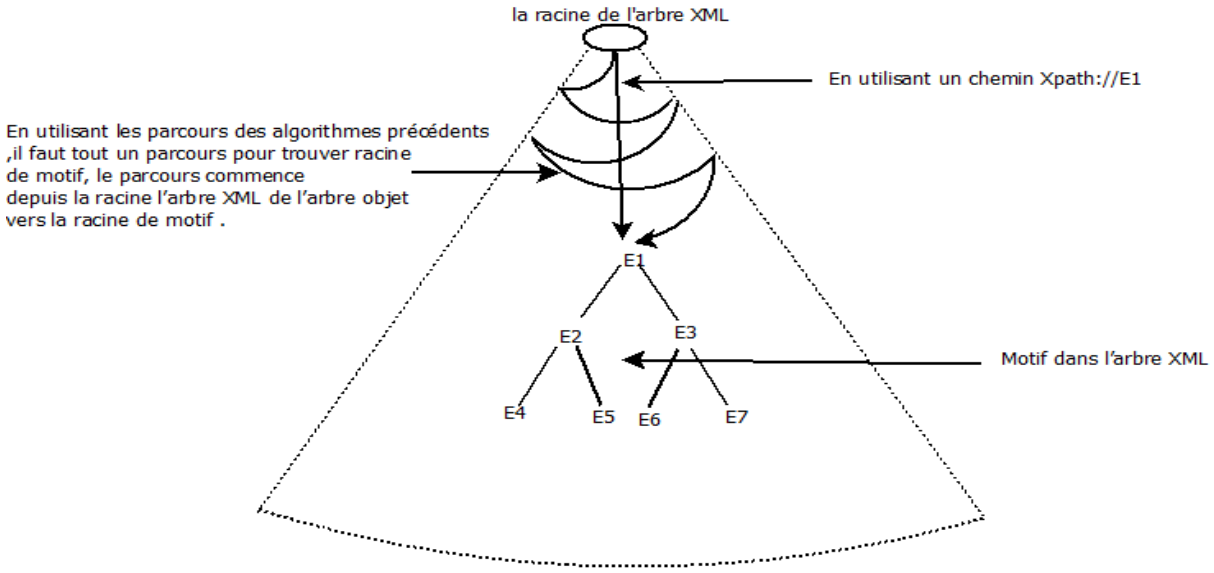


Figure III.14 : la sélection de la racine de motif par l'Xpath

Xpath est un moyen pour optimise la recherche de la racine de motif, parce que Xpath fait la recherche des éléments dans une structure spéciale qui fourni des calculs parallèles de la recherche d'élément XML Voir la figure (Figure II.14**)** .On réalise cette méthode de recherche par le remplacement de la fonction parcours de l'algorithme naïf par un sélecteur de nœud Xpath selectnodes(// la racine de motif) .voir les résultat obtenus dans la partie des teste

4. Teste et discussion sur la recherche de motif XML

Dans cette partie on a réalisé une comparaison entre la recherche de motif avec l'algorithme naïf (qui admet le même principe de déroulement des autres algorithmes de recherche de motif d'arbre) et la nouvelle méthode basé sur l'outil XML (Xpath) et la partie de vérification de motif de l'algorithme naïf voir la figure II.18. On fait des essais sur un ficher représente l'arbre XML et un autre pour représente le motif XML voir les caractéristiques de ces derniers

Figure III.15 : les caractéristiques de l'arbre XML et motif d'arbre XML utiliser

4.1. La recherche de motif exacte

On teste le 1er programme (l'algorithme naïf) et la 2ème programme (l'Xpath et l'algorithme naïf) par le motif exact voir ci-dessous

```
<motif>
  <info>
    <nom>Pillou</nom>
    <prenom>Jean-Francois</prenom>
    <DateN  jour="" mois="" année=""></DateN>
    <lieuN pays="" vile=""></lieuN>
  </info>
</motif>
```

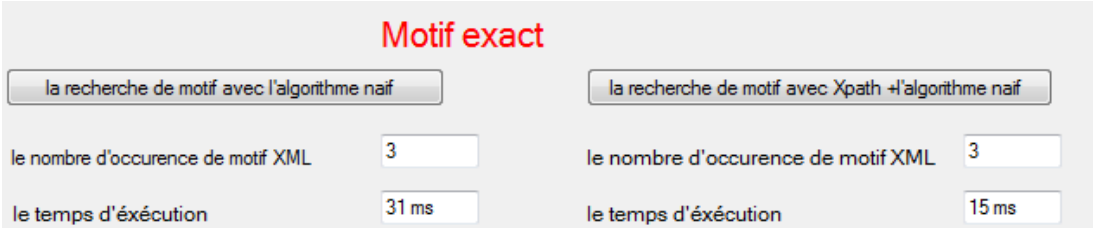


Figure III.16 : le résultat de teste de recherche de motif exacte avec l'algorithme naïf et la recherche de motif avec Xpath et l'algorithme naïf

On a constaté que la nouvelle méthode est plus rapide que la deuxième dans la recherche de motif exacte voir la figure III.16

4.2. La recherche de motif variable

De la même façon on teste pour le motif variable, mais on change seulement le motif par l'ajout des mots réserver pour indique la variance des nœuds, ainsi l'ajoute de chemin xpath pour obtenir le nœud racine les fonctions de calcul voir ci-dessous le motif variable à tester et les résultats

```
<motif a="//motif">
   <motif>cv</motif>
</motif>
```

Figure III.17 : le résultat de teste de recherche de motif variable avec l'algorithme naïf et la recherche de motif avec Xpath et l'algorithme naïf

5. Synthèse

D'après les résultats précédents, dans la RMXML il faut organiser les outils XML et les algorithmes de RM de façon homogène et dans un seul processus pour garantir l'efficacité et l'optimisation de temps dans le futur. Car Si on utilise les RMA de sans l'utilisation de Xpath, on rencontre toujours un parcours de l'arbre XML complet. D'autre part l'outil Xpath ne permet pas la recherche des sous-arbres (motif d'arbre), mais il est rapide pour la sélection des nœuds (occurrence) dans un arbre XML. Il importe doc d'essayer de faire un schéma de processus de la recherche de motif XML en conservant les avantages et en évitant les inconvénients, de Xpath et les algorithmes de RM au futur. Voir la figure II.18 ci-dessous

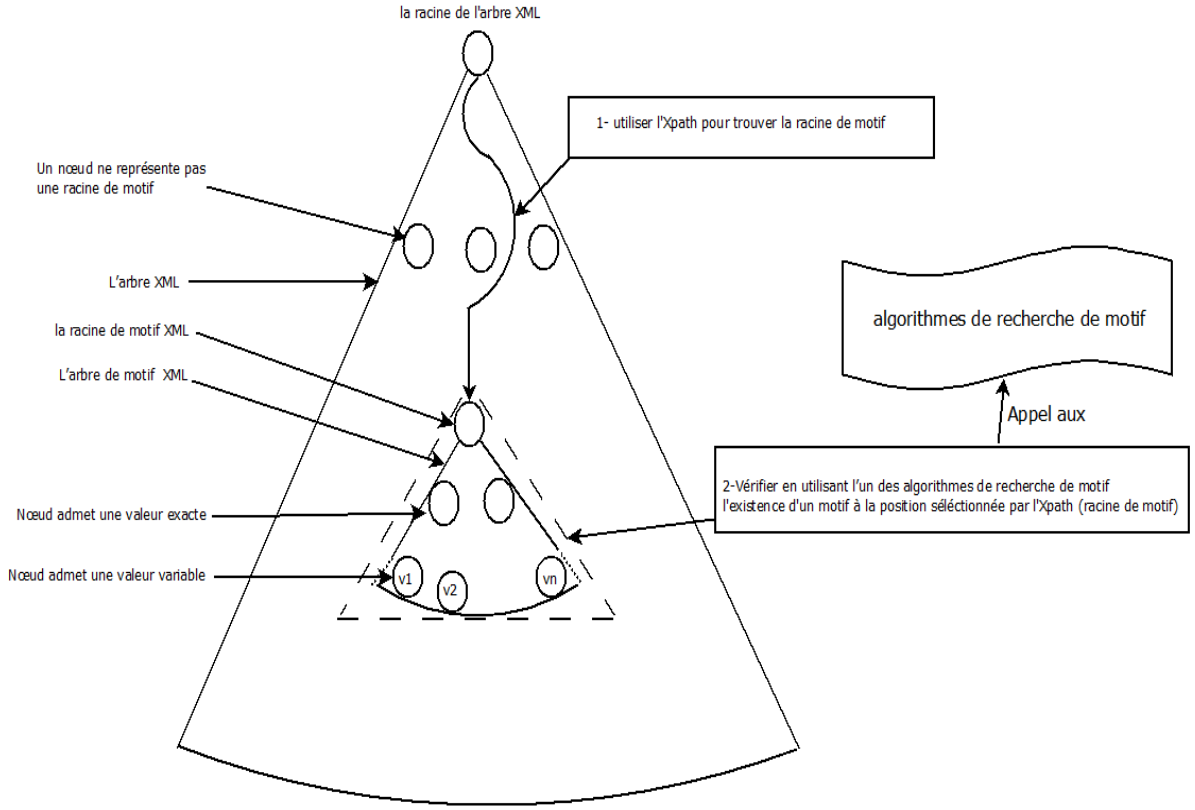


Figure II.18 : Synthèse de la recherche de motif XML

Dans La figure précédente, on lance la RMXML par la recherche de la racine de motif dans arbre XML, en utilisant Xpath ensuite on applique une opération de vérification entre les racines sélectionnées dans l'arbre XML et le motif XML (adaptation entre le sous-arbre XML et le motif XML par l'utilisation des algorithmes de RM). On a ajouté les outils de définition

de structure XML c'est juste pour signaler la possibilité d'utilisation de cet outils dans le processus de la RMXML en futur.

Remarque : Avec ce processus on peut utiliser l'algorithmique parallèle pour assurer une meilleure optimisation, car tout simplement le traitement parallèle dans ce processus est possible, Pour bien illustrer, j'ai essayé de dessiner les étapes de traitement parallèle de ce processus dans la figure II.19 ci-dessous

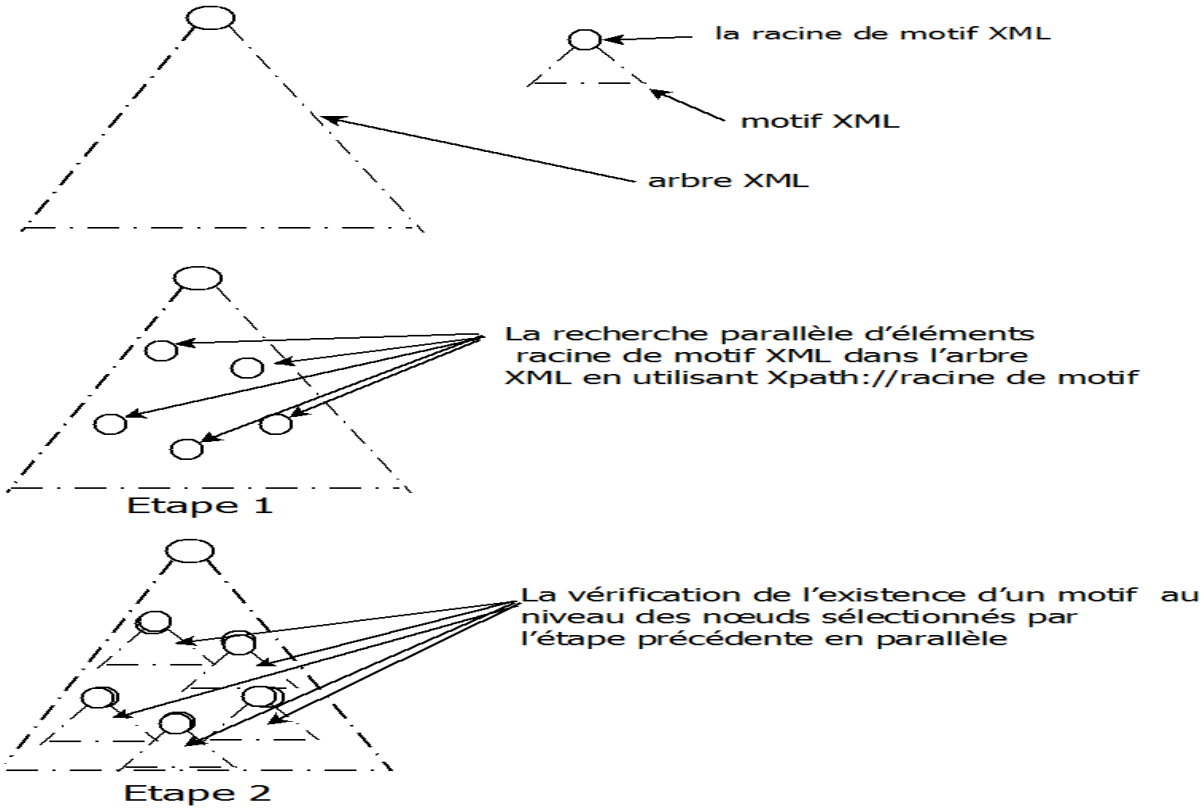


Figure III.19 : la prochaine optimisation de la recherche de motif XML

Conclusion

D'après l'étude théorique et pratique du problème de recherche de motif XML, on a trouvé que l'algorithme naïf de RMA est applicable pour la RMXML exacte et variable. On a aussi essayé d'améliorer ce dernier par l'intégration de Xpath à l'intérieur.

Conclusion générale

Dans ce mémoire nous nous somme intéressé au problème de recherche de motifs dans les documents Xml. En effet, un motif est vu comme un arbre (XML) classique avec des nœuds variables qui peuvent être substitués par n'importe quel contenu.

Nous avons présenté une petite taxonomie sur ce qui existe déjà dans la littérature en termes de recherche de motifs XML et nous avons exposé ses algorithmes d'une manière brève et efficace. Néanmoins, il en existe d'autres algorithmes non cités qui utilisent la notion d'automates d'arbres finis et d'expressions rationnelles d'arbres. Nous souhaitons compléter l'étude de ses algorithmes dans le futur.

Cette étude bibliographique est complétée par la réalisation d'un outil de recherche de motifs dans les documents XML. Bien qu'expérimental, cet outil nous a aidé à prouver beaucoup de propriétés relatives aux algorithmes implémentés comme la preuve de l'importance de Xpath, dans la proposition que nous avons donné bien que la précision de ses estimations ne soit pas tranchante vu l'utilisation d'un simple PC.

Enfin, nous avons touché de près l'un des sujets d'actualité qui est la recherche de motifs dans les documents Xml, nous souhaitons continuer dans cet axe dans les travaux futurs.

Bibliographie

[1] C. Olivier, L'essentiel de XML(Cours XML), Paris: l'Université Diderot, 15/04/2013.

[2] G. Charpentier, «Institut d'électronique et d'informatique Gaspard-Monge (IGM),» [En ligne]. Available: http://www-igm.univ-mlv.fr/~dr/XPOSE2003/xml/contenu_index.htm#plan. [Accès le 23 03 2014].

[3] Laboratoire de Recherche en Informatique, «l'Université Paris-Sud,» [En ligne]. Available: https://www.lri.fr/~benzaken/documents/slxmlsyntax.pdf. [Accès le 30 03 2014].

[4] site enseignement de Jean-Luc Massat, «XML,» [En ligne]. Available: http://jean-luc.massat.perso.luminy.univ-amu.fr/ens/xml/3-dtd.html. [Accès le 01 04 2014].

[5] G. Chagnon, «Ci-gît Gilles,» [En ligne]. Available: http://www.gchagnon.fr/cours/xml/index.html. [Accès le 20 03 2014].

[6] laboratoire lorrain de recherche en informatique et ses applications , «loria.fr,» [En ligne]. Available: http://www.loria.fr/~abelaid/Enseignement/miage-m1/Schema.pdf.

[7] wikipedia, «Espace_de_noms_XML,» [En ligne]. Available: http://fr.wikipedia.org/wiki/Espace_de_noms_XML. [Accès le 05 04 2014].

[8] wikipedia, «Extensible_Stylesheet_Language_Transformations,» [En ligne]. Available: http://fr.wikipedia.org/wiki/Extensible_Stylesheet_Language_Transformations. [Accès le 02 04 2014].

[9] wikipedia, «XQuery,» [En ligne]. Available: http://fr.wikipedia.org/wiki/XQuery.

[10] wikipedia, «XPath,» [En ligne]. Available: http://fr.wikipedia.org/wiki/XPath.

[11] le site web de la MIAGE de Nantes, [En ligne]. Available: http://miage.univ-nantes.fr/miage/D2X1/chapitre_xpath/section_xpath10.htm.

[12] H. Walid-Khaled, «http://hidouci.esi.dz/,» [En ligne]. Available: http://hidouci.esi.dz/algo/cours_arbres.pdf.

[13] L. Cleophas, Tree Algorithms , Two Taxonomies and a Toolkit., TECHNISCHE UNIVERSITEIT EINDHOVEN, Department of Mathematics and Computer Science, 2008.

[14] E. DONALD, J. H. KNUTH, J. MORRIS, R. VAUGHAN et PRATT, «FAST PATTERN MATCHING IN STRINGS*,» *SIAM Journal of Computing,* n° %16, pp. 323-349, 1977.

[15] A. Alfred V et C. Margaret J, « Efficient string matching,» *An aid to bibliographic search ACM,* vol. 18, n° %16, pp. 333-340, June 1975.

[16] H. Christoph M et O. Michael J, « Pattern».

[17] SyncRO Soft SRL, «oxygenxml,» [En ligne]. Available: http://www.oxygenxml.com/index.html. [Accès le 15 03 2014].

[18] wikipedia, «Visual_Basic,» [En ligne]. Available: http://fr.wikipedia.org/wiki/Visual_Basic. [Accès le 15 04 2014].

[19] Université de Nantes, «la MIAGE de Nantes,» [En ligne]. Available: http://miage.univ-nantes.fr/miage/D2X1/chapitre_programmer/section_dom_standard.htm. [Accès le 15 03 2014].

[20] INDUSOFT, «Using XML in InduSoft Web Studio,» InduSoft Systems, 2006.

[21] O'Reilly, «O'Reilly XML.com,» [En ligne]. Available: file:///E:/dom%20et%20sax/XML.com.htm. [Accès le 15 03 2014].

[22] W3Schools, «W3Schools.com,» [En ligne]. Available: file:///E:/dom%20et%20sax/XML%20DOM%20name%20Property.htm. [Accès le 16 03 2014].

Annexe : Model Objet Document (DOM)

1. Définition (XML DOM)

XML DOM ou DOM [18] « est un standard proposé par le W3C. DOM signifie "Document Object Model". Comme son nom l'indique, c'est un standard permettant de représenter des documents (HTML ou XML) sous la forme d'une hiérarchie d'objets. Le document chargé en mémoire est structuré en arbre dont les noeuds sont des instances de classes (au sens de la programmation objet) normalisées. Celles-ci proposent de nombreuses méthodes permettant de manipuler le document en mémoire.». Donc DOM est une interface orientée objet qui analyse les données XML dans une arborescence bien définie Pour faire la manipulation de contenu XML (modifier ou supprimer des éléments). DOM exige que les données XML (un document DOM) doivent être stockées dans la mémoire, il y a une autre interface nommée SAX (API simple XML), qui est un ensemble d'API pour accéder et manipuler les documents XML de manière séquentielle, sax est considéré comme un complément de l'interface DOM.

2. Le principe de fonctionnent de XML DOM

XML DOM c'est une bibliothèque à ajouter aux langages de programmation qui permet l'analyse de document texte XML. Pour rendre la manipulation des éléments XML (élément, texte, commentaire …etc.), comme manipulation objets (class), au niveau de chaque élément XML (objet) il y a des méthodes permettant de modéliser tous les traitements possibles pour cet objet.

On appelle ce traitement de XML, la programmation XML (XML script), car il rend les traitements sur XML comme une suite d'appels des procédures. Donc le DOM est un lien entre le développeur et la structure XML (arbre XML). Aujourd'hui DOM est un standard pour la modélisation de l'XML (programmation de xml).Vous trouver le standard(DOM) dans la plupart des langages. Dans le langage de programmation Visuel Basic DOM est fourni par bibliothèque MSXML (Microsoft's XML Core Services).voir la figure [19] qui exprime le fonctionnement de cette Library (bibliothèque)

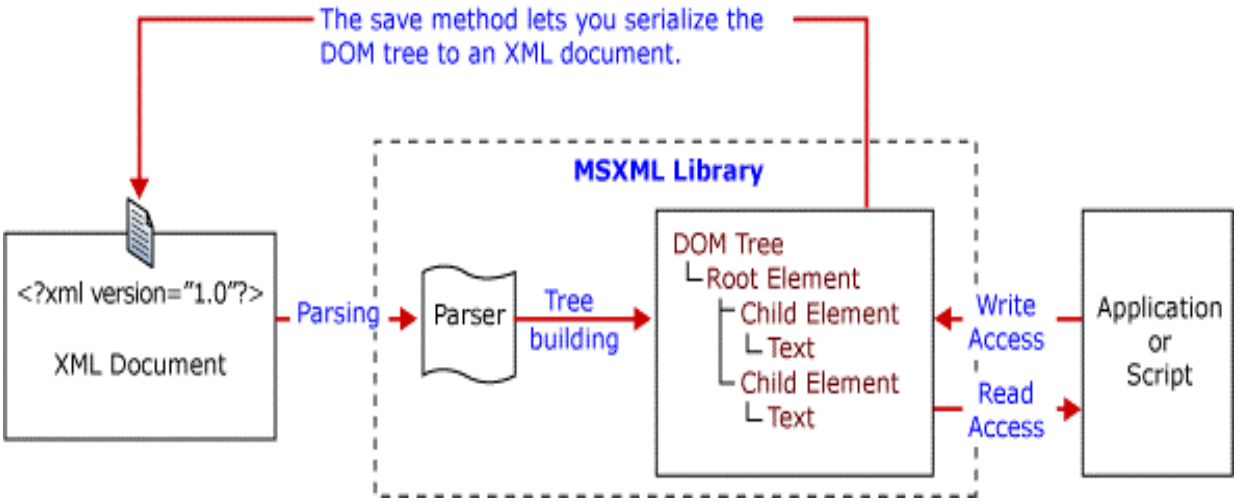


Figure IV.1 : le fonctionnement de MSXML

Dans la figure la bibliothèque réalise deux opérations principales, la 1er c'est la transformation de document texte (forme textuelle) XML en une forme arborescente manipulée par les langages de programmation, la 2ème charge les modifications des langages de programmation dans la structure textuelle de document XML. Tous les éléments qui sont produit lors de la construction de l'arbre DOM d'un document XML sont des nœuds, chaque type de nœud admet des méthodes de manipulations.

3. Les opérations de base de XML DOM

On obtient avec XMLDOM [19] [21] l'arbre DOM d'un document XML. Les éléments dans le document XML sont des nœuds, Qu'il s'agisse d'un élément, attribut, commentaire ou une instruction de traitement. La totalité de document XML (arbre XML) représente l'objet DOMDocument. On accède aux éléments de l'arbre XML à l'aide de l'interface IXMLDOMNode qui admette des méthodes et des propriétés et des interfaces hiérarchiques permettant l'accès aux nœuds-textes , aux nœuds-attributs et aux nœuds-commentaires …etc. pour plus de détails voir la figure IV.2 ci-dessous.

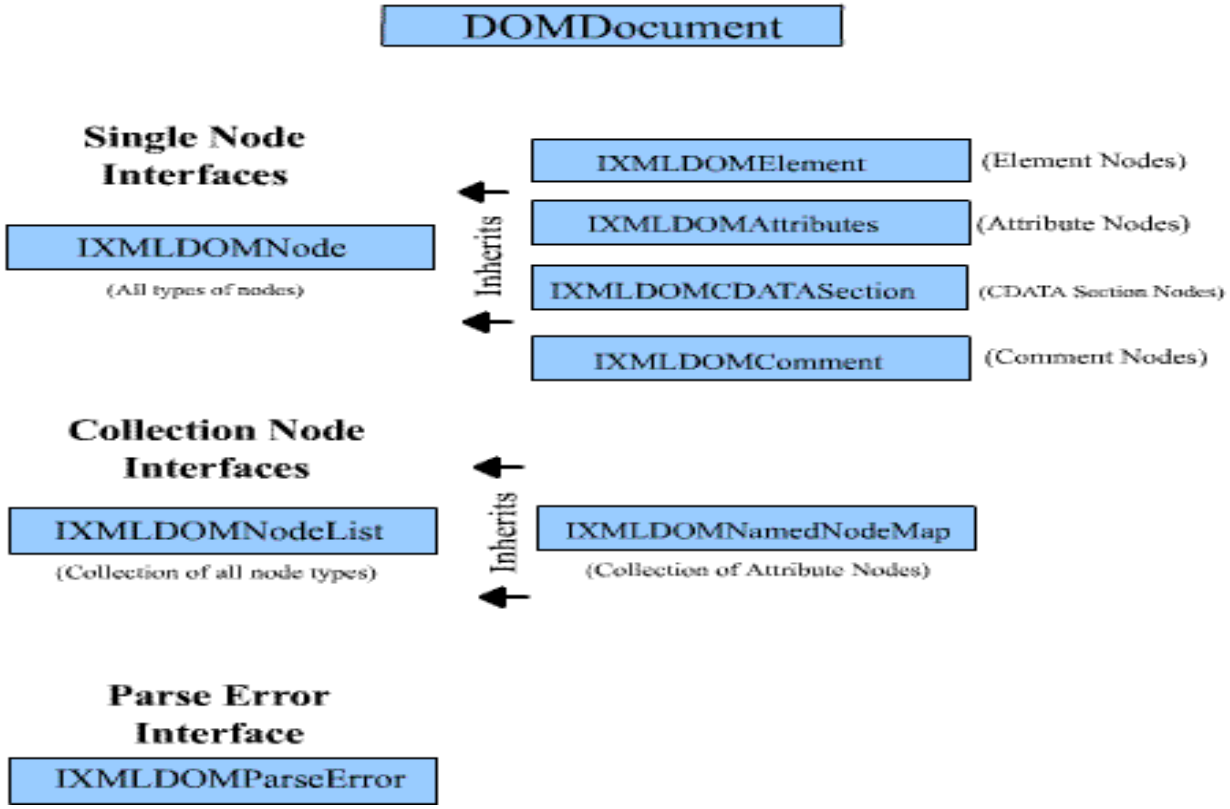


Figure IV.2: MSXML DOMDocument Interfaces

Lorsqu'on utilise les requêtes Xpath , il faut utiliser l'interface IXMLDOMNodeList qui admette une sous-interface pour les attributs IXMLDOMNamedNodeMap, pour stocker les résultats d'une requête(stocker la collection des nœuds ou des attributs sélectionnés par une requête) .Pour chaque interface il y a des méthodes et des propriétés on va résumera les méthodes et les propriétés utilisées dans les opérations de l'application réalisé.

3.1. Lire un document XML et sauvegarder un arbre DOM

Pour lire un document XML en mémoire, il faut initialiser le parseur DOM. Charger le document XML par le code ci-dessous

```
Dim doc As New DOMDocument

doc.load('le chemin vers le fichier XML')
```

Il existe une variante pour charger un document XML en mémoire, si on cherche à charger le string XML il faut utiliser le code ci-dessous.

```
Dim doc As New DOMDocument

doc.loadXML('<a>ajouter le code xml<a/>')
```

Pour sauvegarder un fichier (transformer l'arbre DOM en une forme textuelle) utiliser la méthode Save voir ci-dessous comment faire .

doc .save(chemin de ficher) `'Objet_Document.save(chemin de fichier)'`

3.2. Lire un nœud de l'arbre XML

Il y a plusieurs méthodes pour lire un nœud, la lecture est liée avec la position de nœud et le type de nœud à lire, si on souhaite lire le nœud racine on utilise le code ci-dessous

```
Dim doc As New DOMDocument

Dim racine As IXMLDOMNode

doc.load('le chemin vers le doc XML')

racine = doc.documentElement
```

Si le nœud n'est pas dans la racine il faut utiliser la méthode de cheminement select voir ci-dessous.

```
Dim doc As New DOMDocument

Dim racine As IXMLDOMNode

Dim nouedCH As IXMLDOMNode

Dim nouedCH As IXMLDOMAttribute 'si le résultat est un attribut '

Dim nouedCH As IXMLDOMText       'si le résultat est un texte '

Dim nouedCH As IXMLDOMComment  'si le résultat est un commentaire '

doc.load('le chemin vers le doc XML')

racine = doc.documentElement

noeudCH = racine. selectSingleNode('le chemin vers le noued')
```

Xnode admet toujours un type égale au résultat de requête qui doit être un élément, un attribut, texte ou un commentaire, lorsqu'on utilise des requêtes renvoyant plusieurs nœuds il faut utiliser l'interface type liste nœud pour sauvegarder plusieurs nœuds voir ci-dessous comment faire

```
Dim doc As New DOMDocument

Dim racine As IXMLDOMNode

Dim nouedCH As IXMLDOMNodeList

doc.load('le chemin vers le doc XML')

racine = doc.documentElement
```

```
noeudCH = racine. selectNodes(‘le chemin vers les noeuds’)



        While i < xnodeCH.length

           xnodeCH.item(i)

        End While
```

4. Les méthodes et les propretés de XMLDOM

Pour manipuler l’arbre DOM il suffit d’utiliser les méthodes et les propriétés fournies pour chaque type de nœud, voir dans les tableaux ci-dessous les méthodes et les propriétés les plus utilisées pour chaque type de nœud [19]

Les méthodes	**Description**
appendChild	Ajoute un nouveau nœud enfant après le dernier enfant du nœud
createAttribute	Crée un nouvel attribut avec le nom spécifié par (« »)
createComment	Crée un nœud de commentaire
createElement	Crée un nœud d'élément avec le nom
createTextNode	Crée un nœud de texte contenant les données fournies
hasChildNodes	Vérifier si un nœud a des enfants

insertBefore	Insère un nœud enfant à la gauche du nœud spécifié ou à la fin de la liste.
replaceChild	Remplace le nœud enfant spécifié avec un nouvel enfant
selectNodes	renvoie la liste de nœuds correspondant au chemin
selectSingleNode	renvoie le nœud correspondant au chemin

Tableau IV.1 : Les méthodes de MSXML

Les propretés	**Description**
async	Indique si le téléchargement asynchrone (accès aux fichiers) est autorisé (lecture / écriture),par défaut il est vrai il y a la lecture et écriture dans le fichier
attributes	Retourne une liste des attributs de ce noeud (lecture seule)
childNodes	la liste de nœuds contenant les nœuds enfants (lecture seule)
firstChild	Return le premier enfant d'un noeud (lecture

	seule)
lastChild	Return le dernier enfant d'un noeud (lecture seule)
item	Renvoie l'élément de la collection à l'index spécifié (lecture seule)
length	Indique le nombre d'éléments dans une collection (en lecture seule)
name	Le nom de l'attribut (lecture seule)
next	Renvoie l'élément suivant dans l'objet de collection d'erreur (lecture seule)
nextSibling	Contient le frère suivant du nœud dans la liste d'enfants de parent (lecture seule)
nodeName	Contient le nom qualifié de l'élément, attribut ou référence d'entité, ou une chaîne fixe pour les autres types de nœuds (lecture seule)
nodeType	Indique le type de document XML Object Model (DOM) de nœud, qui détermine des valeurs valides et si le nœud peut avoir des nœuds enfants (lecture seule)

nodeTypedValue	Retourne la valeur du nœud exprimée dans son type de données.
nodeTypeString	Retourne le type de nœud sous forme de chaîne
nodeValue	le texte associé avec le nœud
previousSibling	le frère gauche de ce nœud (lecture seule)
readyState	Indique l'état actuel du document XML (lecture seule)
text	Représente le contenu du texte du nœud ou texte concaténé représentant le nœud et ses descendants (lecture seule)
xml	Contient représentation XML du noeud et tous ses descendants (lecture seule)

Tableau IV.2 : Les propretés de MSXML