REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mouloud MAMMERI – Tizi-Ouzou

Mémoire de fin d'études LMD académique

Domaine Mathématiques et Informatique

Filière Informatique

En vue de l'obtention du diplôme de Master en Informatique

Option : Systèmes Informatiques (SI)

Thème : **Indexation de l'information structurelle de documents XML basée sur les « résumés d'arbres »**

Proposé et dirigé par :

Mr AIT EL HADJ A.

Réalisé par :

TAZIBT Ahmed Amir

LANKRI Hocine

Devant le jury :

Mr HAMADOUCHE D.	**Président**
Mme AOUDJIT R.	**Examinatrice**
Mr DAOUI M.	**Examinateur**
Mr AITELHADJ A.	**Promoteur**

Promotion : 2011

Remerciements

Nous tenons à exprimer notre profonde gratitude à notre promoteur Mr AITELHADJ pour sa patience et ses précieux conseils.

Dédicaces

A nos chers parents qui se sont toujours sacrifiés pour nous, et à nos frères et sœurs qui ont partagé notre vie et à nos amis qui nous témoignent de leur sympathie et de leur soutien.

Résumé

De nos jours, d'énormes volumes d'information circulent sur le Web, et ceci à cause de l'augmentation impressionnante des connaissances de l'humanité ainsi que de l'immense diversité des domaines de ces connaissances. Pour utiliser et suivre cette information l'Homme a recours à des techniques et technologies modernes telles que la Recherche d'Information. La Recherche d'Information peut être considérée comme une passerelle entre l'Homme et l'information. Au cours des années, les corpus documentaires contenant toutes les connaissances ont évolué, et l'utilisateur ne s'est plus intéressé seulement au contenu mais à une nouvelle notion qui est la structure. Cette notion de structure a été introduite par le formalisme des documents semi-structurés, comme les documents XML qui sont les plus répandus sur le Web. Pour effectuer des recherches sur ces documents semi-structurés, les méthodes de recherches d'information classiques ne donnent pas des résultats satisfaisants. Donc, pour remédier à cela, l'indexation structurelle a fait son apparition dans le domaine pour permettre d'avoir des représentations utilisables de ces documents semi-structurés. L'approche d'indexation structurelle traitée dans le cadre de ce mémoire et l'indexation basée sur les « résumés d'arbres » qui représente un réel avancement dans le domaine de la Recherche d'Information car elle garantit des résultats concluants et minimise les contraintes.

Mots clés

Recherche d'Information, Indexation, documents semi-structurés, XML, résumés d'arbres.

Abstract

Nowadays, huge volumes of information circulate on the Web and this because of the impressive increase in knowledge of humanity and of the immense diversity of fields of knowledge. To use and follow this information, man makes use of modern techniques and technologies such as Information Retrieval. Information Retrieval can be seen as a bridge between humans and information. Over the years, the documentary corpus containing all the knowledge has evolved, and the user was no longer interested only in content but a new concept which is the structure. This notion of structure was introduced by the formalism of semi-structured documents, such as XML documents that are most prevalent on the Web. To search the semi-structured documents, research methods of classical information does not give satisfactory results. So to remedy this, the structural indexing has emerged in the field to allow representations to be used in these semi-structured documents. The structural approach to indexing treated as part of this paper and indexing based on "tree summary" which represents areal advancement in the field of Information Retrieval as it ensures conclusive results and minimizes constraints.

Key words

Information Retrieval, indexing, semi-structured documents, XML, tree summary.

Table des matières

Liste des figures

Liste des tableaux

Introduction générale

Nous vivons, de nos jours, dans un monde de technologie et de progrès. L'être humain est assisté dans son quotidien par différentes formes de technologies pour l'accomplissement de ses tâches. L'information qui représente le capital vital de toute société, est également gérée et suivie par divers outils pour permettre de mieux l'utiliser. Car, l'augmentation quasi-exponentielle des connaissances de l'Homme ainsi que leur spécialisation, inévitable, dans des domaines d'intérêt très variés, conduit à la production d'un volume d'information sans précédent. La recherche d'information représente une passerelle entre l'utilisateur et cet univers d'informations qui ne cesse de croître.

Les systèmes de recherche d'information se veulent ainsi comme des messagers dont l'objectif principal est de répondre aux interrogations des utilisateurs. Cependant, comme tout système bâti sur un autre, ils doivent leur efficacité au processus d'indexation qui s'efforce de donner une représentation fidèle, compacte et accessible de l'information recherchée.
Pour améliorer le quotidien de tout un chacun en mettant à sa disposition l'information que souvent, a du mal à retrouver, il faut faire en sorte que les systèmes de recherche d'information soient plus efficaces et plus flexibles. Pour cela l'un des axes auxquels on devrait s'intéresser en premier est l'indexation.

Contexte de travail

Notre travail se situe dans le contexte de la recherche d'information (RI). L'objectif principal des Systèmes de Recherche d'Information (SRI) est de répondre au besoin en information des utilisateurs. Les utilisateurs interrogent, au moyen d'une requête, une base de documents numériques et les SRI leur renvoient une liste de documents susceptibles de répondre à leur besoin.

De nos jours cependant, la nature des sources d'information évolue, et les documents traditionnels « plats » ne contenant que du texte s'enrichissent d'information structurelle et d'information multimédia. Cette évolution est accélérée par l'expansion du Web. De ce fait, les documents structurés ou semi-structurés de type HTML (HyperText Markup Language) ou

XML (eXtensible Markup Language) tendent à former la majorité des documents numériques mis à disposition des utilisateurs.

Les systèmes de recherche d'information doivent tirer profit de cette nouvelle notion qu'est la structure pour mieux servir les utilisateurs. Les méthodes de recherche d'information existantes doivent être adaptées ou de nouvelles méthodes doivent être proposées. C'est dans ce contexte de recherche d'information structurée que se situe plus particulièrement notre travail. Nous nous plaçons plus précisément dans le cadre de documents semi-structurés, c'est à dire de documents ne disposant pas d'une structure fixe et homogène, mais au contraire d'une structure flexible ainsi que de contenus hétérogènes. Nous utiliserons tout au long de ce rapport le format XML pour illustrer les propos de notre recherche et nos exemples.

Problématique et solution préconisée

Un document XML est caractérisé par du contenu (texte) et une structure (balises). Ce type de documents ne peut cependant être exploité efficacement par les méthodes classiques de Recherche d'Information. En effet, ces dernières ne traitent un document que du point de vue de son contenu, alors que le format XML ajoute un nouveau paramètre qui est la structure (balises). Ceci impose alors d'adapter les méthodes classiques de RI à cette structure ou d'introduire de nouvelles méthodes afin d'exploiter au mieux les informations disponibles sous le format XML.

Une fonctionnalité essentielle a été introduite dans ce but, c'est la notion d'indexation qui consiste à extraire les mots clés de recherche des documents. Dans le cas des documents textes plats, le contenu textuel des documents est traité afin de trouver et de pondérer les termes les plus significatifs des documents. Cependant, dans le cas des documents semi-structurés, la dimension structurelle s'ajoute au contenu. Ceci nous mène à nous poser la question suivante : quelle est l'approche d'indexation « structurelle » la mieux adaptée et la plus efficace ?

Le travail que nous avons entrepris rentre dans ce carde et il est motivé par l'extension des méthodes actuelles de RI pour les adapter à la notion de structure. Dans ce but nous préconisons une approche qui consiste au préalable à représenter chaque document XML par

sa structure générique (ou résumé d'arbre), et c'est cette structure de « résumé d'arbre » qui sera utilisée pour réaliser une indexation structurelle.

Ce choix de représenter un document XML par son « arbre résumé » résulte du fait que ce type de représentation apporte une multitude d'avantages tels que la rapidité des traitements, l'efficacité dans le retour des résultats et la fiabilité de ces résultats. Nous verrons ces aspects plus en détail dans le développement de ce mémoire.

La classification structurelle est la finalité de cette indexation structurelle et a pour rôle de regrouper des documents XML structurellement similaires dans des clusters (ou classes), afin de réduire le temps de réponse et d'augmenter la précision des moteurs de recherche. Ceci n'entre pas dans le cadre de notre travail, mais peut constituer un débouché.

Organisation du mémoire

Ce mémoire est constitué, après cette introduction, de quatre chapitres. Le premier chapitre s'intéresse au langage XML ainsi qu'aux différents concepts qui s'y rattachent comme le SGML, le HTML, ou même les notions de structure, de DTD et de parseurs qui rentrent dans le cadre de ce langage. Ensuite, le second chapitre s'étale sur l'indexation de l'information structurelle dans le cas de documents XML, en donnant des définitions préliminaires importantes, puis en passant par les différentes méthodes d'indexation dans le but d'interroger une collection de documents XML et celles dans le but de classifier des documents. Le troisième chapitre s'intéresse de manière détaillée aux approches d'extraction de résumés d'arbres, et plus précisément à celle utilisée dans le cadre de notre travail de programmation. Dans le quatrième chapitre, nous avons plongé dans le vif du sujet de notre travail en expliquant notre vision de programmation de l'approche d'extraction de résumés d'arbres, et en détaillant les différentes procédures que l'on a programmé, tout en montrant toutes les interactions de notre application par des exemples concrets et bien illustrés, et l'intérêt de notre contribution. Enfin, ce mémoire se clôture avec une conclusion générale.

Chapitre I

Le langage XML

Introduction

Les langages de balisage (sous-classe des langages de description) représentent une classe de langages spécialisés dans l'enrichissement d'information textuelle. Ils opèrent grâce aux balises, unités sémantiques délimitant chacune un ensemble à l'intérieur d'un fichier texte.

L'inclusion de balises permet de transférer à la fois la structure du document et son contenu. Cette structure est compréhensible par un programme informatique, ce qui autorise un affichage personnalisé selon des règles préétablies.

Historiquement, les langages de balisage les plus utilisés sur le Web sont des applications dérivées de SGML, parmi ces langages on a HTML qui est un langage de balisage destiné exclusivement au World Wide Web.

Le langage XML dérive également de SGML, et hérite des avantages de ce dernier, ainsi que des avantages du HTML.

XML est l'abréviation de « Extensible Markup Language ». Contrairement à ce qui est souvent dit, XML n'est pas un langage de programmation. On ne peut pas faire de tests, ni inclure un fichier dans un autre. En très gros, XML sert uniquement à stocker des données.

XML est simplement une méthode pour représenter les données. Celles-ci sont écrites entre des balises ou sous forme d'attributs, et l'ensemble est écrit sous forme d'un arbre.
Dans ce chapitre, nous allons détailler brièvement les notions de SGML, HTML et nous détaillerons tout ce qui gravite autour du XML.

1.1. Le SGML

SGML est un langage de codage de données dont l'objectif est de permettre, dans un échange entre systèmes informatiques, de transférer, en même temps, des données textuelles et leurs structures. SGML était très utilisé avant qu'XML n'existe : c'est d'ailleurs à partir de ce langage (et en tenant compte de ses différentes utilisations) qu'a été élaboré le standard XML.

On a dit beaucoup de choses sur SGML : que le langage était lourd, que les implémentations étaient difficiles, etc. De fait, ce n'était pas la technologie qui était lourde, mais les applications elles-mêmes. En effet, s'il est nécessaire de manipuler un manuel de maintenance aéronautique de quelques centaines de méga-octets, avec des arbres d'une profondeur allant jusqu'à des dizaines de niveaux, la technologie n'y fait rien car les manipulations sont de toutes les façons d’une complexité élevée.

En termes d’avenir, d'un point de vue technique, SGML peut sans problème être remplacé par XML : tout ce qui est nécessaire à SGML existe à peu près dans XML. D'un point de vue pratique, les utilisateurs de SGML y gagneront en diversité d'outils utilisables et apprécieront à son juste avantage la prise en compte par XML d'Unicode XML.

À la différence d'XML, un document SGML doit toujours être valide au regard d'une DTD (Document Type Definition). Les concepteurs, en raison de leur orientation documentaire, s'intéressaient beaucoup à la notion de validation électronique et automatique. Cette validation au regard d'une DTD permet de s'assurer que l'information est saisie conformément au modèle et que l'on peut donc lui appliquer les traitements automatisés sans avoir à valider la structure du document au préalable. A noter que cela reste vrai pour XML mais que cette dernière recommandation supprime l'obligation de validation pour lecture car des processus d'assurance qualité peuvent remplacer cette validation à toutes les étapes d'un processus informatique.

1.2. Le HTML

Le **HTML** (*HyperText Mark-Up Language*) est un langage dit de marquage (de structuration ou de balisage) dont le rôle est de formaliser l'écriture d'un document avec des balises de formatage. Les balises permettent d'indiquer la façon dont doit être présenté le document et les liens qu'il établit avec d'autres documents.

Il est important de comprendre que le langage HTML est un standard, c'est-à-dire qu'il s'agit de recommandations publiées par un consortium international : le World Wide Web Consortium (**W3C**).

Les spécifications officielles du HTML décrivent donc les "instructions" HTML mais en aucun cas leur implémentation, c'est-à-dire leur traduction en programmes d'ordinateur, afin de permettre la consultation de pages web indépendamment du système d'exploitation ou de l'architecture de l'ordinateur.

Toutefois, aussi étoffées les spécifications soient-elles, il existe toujours une marge d'interprétation de la part des navigateurs, ce qui explique qu'une même page web puisse s'afficher différemment d'un navigateur Internet à l'autre.

De plus, il arrive parfois que certains éditeurs de logiciels ajoutent des instructions HTML propriétaires, c'est-à-dire ne faisant pas partie des spécifications du W3C. Ainsi des pages web contenant ce type d'instruction pourront être parfaitement affichées sur un navigateur et seront totalement ou en partie illisibles sur les autres.

Le HTML est destiné presque exclusivement au Web (navigateurs internet avec protocole HTTP) à la différence de XML qui est destiné à l'échange de données hiérarchisées entre systèmes informatiques.

XML est plus souple et plus personnalisable en fonction des besoins, car on peut définir de nouvelles balises pour un besoin particulier ; au contraire, le schéma du HTML est fixe on se doit d'utiliser les balises définies par le W3C. A cause de cela, HTML est considéré comme une norme mourante par le W3C.

1.3. Le XML

XML (*Extensible Markup Language*, « langage de balisage extensible ») est un langage informatique de balisage générique. Il sert essentiellement à stocker/transférer des données de type texte Unicode structuré en champs arborescents. Le W3C, promoteur de standards favorisant l'échange d'informations sur Internet, recommande la syntaxe XML pour exprimer des langages de balisages spécifiques. De nombreux langages respectent la syntaxe XML : XHTML, SVG, XSLT, etc.

1.3.1. Le document XML

On recense deux types de documents XML : les fichiers orientés documents et les fichiers orientés données [3].

Lorsque les données sont élaborées par des êtres humains, on dit que les fichiers XML produits sont orientés document. Lorsque les données sont construites automatiquement par des programmes, on dit que les fichiers XML sont orientés données. Un fichier XML orienté document peut être, par exemple, un livre, un article, un message, etc.

Un fichier XML orienté donnée est, par exemple, un sous-ensemble d'une base de données.

Il faut noter que l'élaboration des fichiers XML nécessite des moyens de contrôle et d'édition plus ou moins sophistiqués. On n'utilisera pas pour fabriquer un ouvrage en XML un éditeur trop rudimentaire (comme le bloc-notes sous l'environnement Windows). L'édition des documents XML sera abordée plus loin dans ce chapitre.

1.3.2. Structure d'un document XML

Un document XML est structuré en 3 parties :

La première partie, appelée prologue permet d'indiquer la version de la norme XML utilisée pour créer le document (cette indication est obligatoire) ainsi que le jeu de caractères (en anglais encoding) utilisé dans le document (attribut facultatif, on spécifie qu'il s'agit du jeu ISO-8859-1, jeu LATIN, pour permettre de prendre en compte les accents français). Ainsi le prologue est une ligne du type :

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

Le prologue se poursuit avec des informations facultatives sur des instructions de traitement à destination d'applications particulières. Leur syntaxe est la suivante :

<?instruction de traitement?>

Le second élément est une déclaration de type de document (à l'aide d'un fichier annexe appelé DTD, que l'on détaillera plus loin dans ce chapitre)

Et enfin la dernière composante d'un fichier XML est le corps du document XML.

Le corps d'un document XML consiste essentiellement en un arbre d'éléments. La racine de cet arbre correspond à l'élément racine (Document Elément) du document. Les autres nœuds de l'arbre correspondent aux éléments descendants de l'élément racine du document.

Les éléments d'un document XML possèdent des relations de type (parent, enfant). Les parties de l'arbre, possédant un enfant sont nommées branches, tandis que les parties dépourvues d'enfants, sont nommées feuilles ou nœuds terminaux. Chaque document ne possède qu'un seul élément racine qui contient tous les autres [3]. Par exemple, la Figure 1.1 donne une idée sur la structure logique d'un document XML.

```
<doc>
  <parent>
    <nom>Ceci est</nom>
  du texte dans mon element
  </parent>
</doc>
```

Figure 1.1 – Structure arborescente d'un document XML

En général, les éléments XML peuvent contenir du texte et d'autres éléments qui sont leurs sous-éléments ou leurs descendants. Les éléments sont décrits comme ayant un contenu mixte. En effet, sur le document de l'exemple de la Figure 1.1 <parent> possède deux descendants :

– Un descendant qui est la balise <nom> ;

– Un autre descendant contenant le texte «du texte dans mon élément ».

On peut restructurer le document de la Figure 1.1 à notre guise, en introduisant des attributs comme sur la Figure 1.2.

Les attributs apportent des informations sur l'élément qui les contient (ils ne sont pas appropriés pour contenir des données). Il n'y a pas de limite sur le nombre d'attributs utilisables dans un élément ; il faut cependant trouver un compromis entre utilisation d'attributs ou emploi d'un nouvel élément et garder à l'esprit que les éléments décrivent les données alors que les attributs définissent les éléments qui les contiennent.

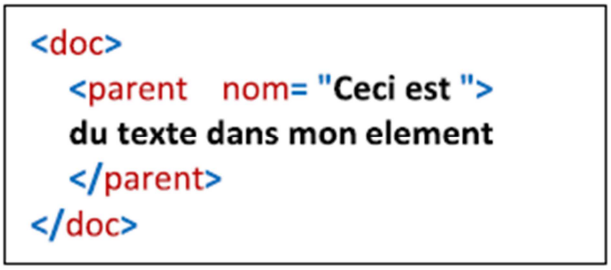


Figure 1.2 – Document XML avec attributs

Notons que le document de la Figure 1.2 est sémantiquement équivalent a celui de la Figure 1.1, ce qui a changé, c'est uniquement la conception de sa structure. Cette flexibilité dans la manière de structurer un document est une richesse qui permet aux utilisateurs de XML, de créer leurs documents selon leur préférence.

En fait, le format XML permet de produire aussi bien des documents structurés que des documents semi-structurés.

– Les documents structures possèdent une structure régulière, c'est à dire ne contiennent pas d'éléments mixtes, et l'ordre des éléments qu'ils contiennent est généralement indifférent.

– Les documents semi-structures possèdent une structure flexible et des contenus hétérogènes. La mise à jour d'une donnée entraine une modification de la structure de l'ensemble.

1.3.3. Validation d'un document XML

On associe à un document XML un schéma, qui peut être vu comme le schéma d'une base de données relationnelle. La validation d'un document XML garantit que la structure de données utilisée respecte ce schéma. On peut faire l'analogie avec le respect des règles d'orthographe et de grammaire d'une langue. Les documents XML qui circulent doivent ainsi être en accord avec ce schéma pour être acceptés par la plate-forme. Dans le cas contraire ils sont rejetés et doivent être refaits.

Lorsque les flux d'échanges sont denses, la validation peut présenter pour inconvénient de consommer des ressources. Il est difficile de raisonner pour tous les cas, mais la validation peut être considérée comme incontournable à certaines étapes de préparation du cadre d'exploitation. Lorsque les flux sont considérés comme stables, il est alors possible de pratiquer une forme d'assouplissement des règles dans l'optique d'améliorer les performances.

1.3.4. L'édition d'un document XML

L'édition de document XML peut prendre diverses formes, notamment en fonction de sa finalité.

❖ Cas des formats orientés document

Pour réaliser un ouvrage, un article… en XML il n'est pas conseillé d'utiliser un éditeur de texte quelconque. La réalisation de tels documents impose de se focaliser sur le contenu et non sur la syntaxe du format de document. Pour arriver à alléger la part de ce travail, il existe des outils qui proposent l'édition en WYSIWYG (what you see is what you get) : l'auteur n'a alors plus l'impression de réaliser un document XML mais simplement d'utiliser un éditeur graphique (comme MS Word ou OpenOffice.org). Ces outils utilisent souvent une feuille de styles CSS (Cascading Style Sheets) qui donne une représentation graphique à telles ou telles parties du document XML. C'est pourquoi, certains logiciels proposent une édition XML via un navigateur de type Mozilla Firefox ou Internet Explorer.

❖ Cas des formats orientés données

Dans ce type de format, il n'y a pas de représentation facilement utilisable pour l'être humain, l'idéal étant de passer par une application qui masquera la localisation des données.

- ✓ Édition avec un formulaire

Certaines solutions visent à analyser les schémas des fichiers XML pour générer un formulaire de saisie. Cela peut être intéressant lorsque ce formulaire est disponible via un navigateur. Parmi les éditeurs proposant cette solution, citons EditLive! et Microsoft, avec InfoPath.

- ✓ Éditeurs plus généralistes

Les éditeurs généralistes sont une autre forme d'éditeurs qui s'adressent plutôt à des techniciens. Il existe de nombreux produits, qui offrent tous la validation et la transformation. Ils se démarquent par certaines facilités. Le plus connu est l'éditeur XMLSpy.

1.3.5. XML et DTD

XML permet d'utiliser un fichier afin de vérifier qu'un document XML est conforme à une syntaxe donnée. La norme XML définit ainsi une définition de document type appelée DTD (*Document Type Definition*), c'est-à-dire une grammaire permettant de vérifier la conformité du document XML. La norme XML n'impose pas l'utilisation d'une DTD pour un document XML, mais elle impose par contre le respect exact des règles de base de la norme XML.

Une DTD (*Document Type Definition*) est une forme de grammaire relativement ancienne car issue de l'univers SGML (*Standard Generalized Markup Language*). Elle a l'avantage d'être rapide à écrire, tout en présentant l'inconvénient d'être pauvre en possibilités de contrôle (typage de données, par exemple). Autre point négatif, les espaces de noms sont difficilement gérables car il faut intégrer, dans la grammaire, les préfixes, ce qui est contraire à l'esprit des espaces de noms.

On peut distinguer selon l'utilisation ou non de DTD deux types de documents XML :

- **Document valide** pour un document XML comportant une DTD.
- **Document bien formé** pour un document XML ne comportant pas de DTD mais répondant aux règles de base du XML.

Une DTD peut être définie de deux façons :

- Sous **forme interne**, c'est-à-dire en incluant la grammaire au sein même du document.
- Sous **forme externe**, soit en appelant un fichier contenant la grammaire à partir d'un fichier local ou bien en y accédant par son URL. L'usage voudra que l'on privilégie cette forme pour des raisons de maintenance et de facilité d'accès.

1.3.6. Les parseurs XML

L'analyseur syntaxique (généralement francisé en *parseur*) est un outil logiciel permettant de parcourir un document et d'en extraire des informations. Dans le cas de XML, on parlera de parseur XML.

Le parseur est l'élément de programmation le plus important, puisque c'est lui qui réalise le travail d'analyse du document XML. Son rôle est de vérifier la cohérence du document XML (en termes syntaxique et/ou par rapport à un schéma ou une DTD) et de transmettre à l'application les informations utiles au traitement du document.

Pour une application, un parseur fait partie d'une API ou d'une bibliothèque : ce n'est pas un organe indépendant mais un bloc de traitement que l'on peut contrôler, comme n'importe quelle méthode externe.
Un parseur XML permet de créer une structure hiérarchique contenant les données contenues dans le document XML.

On distingue deux types de parseurs XML :

- les parseurs validants (validating) permettant de vérifier qu'un document XML est conforme à sa DTD.
- les parseurs non validants (non-validating) se contentant de vérifier que le document XML est bien formé (c'est-à-dire respectant la syntaxe XML de base).

Les analyseurs XML sont également divisés selon l'approche qu'ils utilisent pour traiter le document. On distingue actuellement deux types d'approches :

- Les API utilisant une approche **hiérarchique** : les analyseurs utilisant cette technique construisent une structure hiérarchique contenant des objets représentant les éléments du document, et dont les méthodes permettent d'accéder aux propriétés. La principale API utilisant cette approche est **DOM (Document Object Model)**.
- Les API basés sur un mode **événementiel** permettent de réagir à des événements (comme le début d'un élément, la fin d'un élément, etc.) et de renvoyer le résultat à l'application utilisant cette API. **SAX (Simple API for XML)** est la principale interface utilisant l'aspect événementiel.

Ainsi, on tend à associer l'approche hiérarchique avec **DOM** et l'approche événementielle avec **SAX**.

1.3.7. Document Object Model (DOM)

Le **Document Object Model** (ou **DOM**) est une recommandation du W3C qui décrit une interface indépendante de tout langage de programmation et de toute plate-forme, permettant à des programmes informatiques et à des scripts d'accéder ou de mettre à jour le contenu, la structure ou le style de documents XML. Le document peut ensuite être traité et les résultats de ces traitements peuvent être réincorporés dans le document tel qu'il sera présenté.

DOM permet de construire une arborescence de la structure d'un document et de ses éléments. Il est donc préférable de parcourir et de mémoriser l'intégralité du document avant de pouvoir effectuer les traitements voulus. Pour cette raison, les programmes utilisant DOM ont souvent une empreinte mémoire volumineuse en cours de traitement. À l'inverse, à partir d'un arbre DOM donné, il est possible de générer des documents dans le langage de balisage voulu, qui pourront à leur tour être manipulés par l'interface DOM.

DOM est utilisé pour pouvoir modifier facilement des documents XML ou accéder au contenu des pages web.

DOM se divise en deux spécifications :

- La spécification **DOM level 1** (DOM niveau 1) se séparant en deux catégories
    - *Core DOM level 1*: La spécification pour les documents en général (dont XML).
    - *HTML DOM level 1*: La spécification retenant uniquement les méthodes applicables à HTML.
- La spécificaion **DOM level 2** ajoutant de nouvelles fonctionnalités comme la prise en compte des feuilles de style CSS dans la hiérarchie d'objets.

1.3.8. Simple API for XML (SAX)

SAX (*Simple API for XML*) est une API basée sur un modèle événementiel, qui transforme un document XML en un flux d'évènements déclenchés par la lecture d'éléments syntaxiques XML (balise ouvrante, balise fermante, etc.). Le modèle est quelque peu calqué sur celui des interfaces graphiques, l'application cliente devenant un "écouteur d'événement".

Plutôt que de représenter la totalité du document XML en mémoire, le parseur réalise un découpage du document en petites unités et transmet ces unités à l'application au fur et à mesure de l'analyse du document. Une unité représentera, par exemple, l'ouverture d'un élément ou sa fermeture, la rencontre d'un texte… On le comprendra, dans ce système, l'application n'a pas de représentation globale du document ou plutôt le parseur ne fournit qu'un cheminement dans le document, que l'application est libre de stocker ou non.

Les avantages de SAX sont bien sûr liés à la vitesse de traitement et à la faible consommation mémoire côté parseur. On le retiendra sur des traitements de documents de taille importante. La contrepartie est l'accroissement de la complexité de la programmation et l'impossibilité de pratiquer des requêtes par des fonctions telles que XPath ou XQuery, qui nécessitent une représentation globale en mémoire.

Pour définir les actions effectuées par le parseur, SAX se compose de quatre interfaces appelées par le parseur pour les différents événements rencontrés durant le parsing. Ces interfaces sont définies dans la Figure 1.3 suivante.

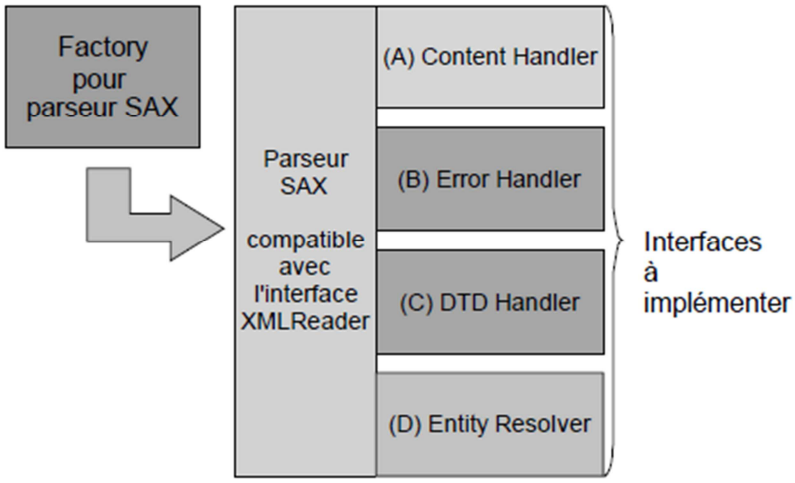


Figure 1.3 - Les différentes parties de SAX.

ErrorHandler

L'interface ErrorHandler permet aux applications de gérer les erreurs rencontrées lors du parsing. Elle permet, par exemple, d'intercepter des erreurs dues à un document XML mal formé que l'on voudrait ignorer.

DTDHandler

Cette interface sert à résoudre les notations et les entités définies dans la DTD associée au document XML analysé.

EntityResolver

Cette interface sert à résoudre les entités externes contenues dans le document XML. Quand le parseur rencontre une entité externe, il appelle la méthode resolveEntity écrite par le développeur.

ContentHandler

Content Handler est l'interface centrale, car c'est elle qui sert à la communication avec l'application. Son rôle est de transmettre les événements du parseur à l'application. Elle comporte autant de méthodes que de formes de représentation XML (ouverture et fermeture des éléments, espaces de noms, textes, commentaire, etc.).
Voici un extrait de ces méthodes, sachant que nous avons choisi une représentation Java qui est facilement transposable dans divers langages :

- Réception du texte :

  Package org.xml.sax :

  *void**characters**(char[] ch, int start, int length)*
- Notification de fin de document :

  *void**endDocument**()*
- Notification de fin de balise :

  *void**endElement**(java.lang.Stringuri, java.lang.StringlocalName, java.lang.StringqName)*
- Notification de fin de portée d'un préfixe :

  *void**endPrefixMapping**(java.lang.String prefix)*
- Notification de blancs ignorés :

  *void**ignorableWhitespace**(char[] ch, int start, int length)*

- Notification d’une instruction de traitement :

  *void**processingInstruction**(java.lang.String target, java.lang.String data)*
- Objet utilisé pour la localisation du parsing (colonne, ligne) :

  *void**setDocumentLocator**(Locator locator)*
- Entité ignorée :

  *void**skippedEntity**(java.lang.String name)*
- Notification de début du document :

  void**startDocument**()
- Notification de début de balise :

  void**startElement**(java.lang.Stringuri, java.lang.StringlocalName,
  java.lang.StringqName, Attributesatts)
- Rapporte la déclaration d’un préfixe pour un espace de noms :

  *public void **startPrefixMapping**(String prefix, String uri)*

Conclusion

XML est plus qu'un simple format texte pour décrire des documents. Il s'agit d'un mécanisme pour décrire les données structurées et semi-structurées, ce qui permet d'accéder à une riche famille de technologies pour le traitement de telles données.

XML est le digne héritier de SGML, et hérite des avantages de ce dernier ainsi que du HTML.

Dans ce chapitre nous avons donné un aperçu de XML et de toutes les technologies qui gravitent autour. Dans le chapitre suivant nous nous intéresserons à l’indexation de l’Information Structurelle de documents XML.

Chapitre II

Indexation de l'information structurelle de documents XML

Introduction

Les SRI (Systèmes de Recherche d'Information) classiques traitent en général des ensembles de documents plats, s'intéressant uniquement au contenu, et laissant complètement de côté la structure et les liens pouvant éventuellement exister entre les documents. Cependant, avec l'évolution des corpus de documents, notamment avec la propagation et l'émergence des données semi-structurées sur le Web, les choses ont pris une autre tournure. En effet, les documents XML grâce à leur format, obligent les SRI à tenir compte aussi bien du contenu que de la structure afin de cibler avec une précision plus importante et avec plus d'exhaustivité les informations spécifiques dont l'utilisateur a besoin.

La composante structurelle peut servir à établir une classification structurelle de ces documents. Cette dernière sert à améliorer et optimiser le processus de recherche d'information. Autrement dit, au lieu de se focaliser, comme pour les SRI standards, sur une recherche d'information sur de grands nombres de documents (sacs de mots), une classification structurelle optimisera cette recherche et l'orientera vers une certaine classe affinée de documents. On y gagne en réduisant significativement le nombre de documents à traiter, et aussi en regroupant des documents qui sont similaires structurellement, et on pourra avoir plus de documents pertinents retournés par le moteur de recherche.

L'indexation proprement dite peut prendre deux tournures différentes : l'indexation orientée classification (documents) et l'indexation orientée interrogation (requête). En bref, l'objectif de la première approche est de résumer ou de présenter le contenu de chaque document en fournissant un maximum d'informations sur ce dernier. Quant à la seconde approche, elle sert à refléter, pour chaque document, les requêtes d'interrogation pour

lesquelles il est pertinent en confrontant chaque document du corpus documentaire à une liste de requêtes prédéfinies.

Dans notre travail, nous nous concentrerons plus sur les approches orientées classification en passant brièvement sur des approches orientées interrogation, donc dans ce chapitre nous allons donner plus de détails sur le processus d'indexation puis nous passerons aux méthodes d'indexation de l'information structurelle de documents XML.

2.1. Le processus d'indexation

Selon les auteurs [2] [9], le processus d'indexation est mis en œuvre afin d'extraire préalablement une représentation homogène du contenu sémantique, sous forme de termes d'indexation qui sont des éléments d'un langage d'indexation. Dans un SRI un document est considéré comme un support qui véhicule de l'information. La phase d'indexation permet donc, de capturer cette information et de la représenter selon un modèle: le modèle de documents. L'information capturée correspond au contenu sémantique du document et la représentation de ce contenu sémantique est appelée un index de document. L'indexation joue un rôle primordial dans la construction d'un SRI. Sa finalité est d'affiner les procédures d'accès aux bases documentaires. L'indexation peut être :

- Manuelle : chaque document est analysé par un spécialiste du domaine ou par un documentaliste ;
- Automatique : le processus d'indexation est entièrement informatisé et automatisé ;
- Semi-automatique : le choix final revient au spécialiste ou au documentaliste, qui intervient souvent pour choisir d'autres termes plus significatifs.

L'indexation manuelle permet d'avoir des résultats plus précis et plus pertinents retournés par le SRI, mais elle présente plusieurs inconvénients : Deux indexeurs différents peuvent présenter des termes totalement différents pour caractériser un même document, et à deux moments différents, un même indexeur peut présenter deux termes différents pour caractériser un même mot-clé. De plus, cette méthode engendre une grande perte de ressource temporelle.
En ce qui concerne l'indexation semi-automatique, les indexeurs utilisent une base terminologique (liste organisée de descripteurs ou mots clés) obéissant à des règles spécifiques et reliés entre eux par des règles sémantiques.
L'indexation automatique utilise un ensemble de traitements automatisés sur un document. Parmi ceux-là :

- L'extraction automatique des mots clés du document ;

- L’élimination des mots « vides » ;
- La normalisation des mots-clés du document ;
- La détection des groupes de mots-clés ;
- La pondération des mots-clés ;
- La création de l’index.

Le choix d’une méthode par rapport aux autres peut dépendre de plusieurs paramètres, dont le plus déterminant est le volume des collections documentaires.

Il existe plusieurs études comparatives de ces méthodes, et le résultat de ces dernières montre que les avantages et les inconvénients de chacune des approches s’équilibrent : le choix d’une méthode dépend du domaine, de la collection et de l’application considérée.

En bref, l’indexation constitue une représentation du document et de son contenu dans le but de faciliter son exploitation, on distingue à ce titre deux types d’indexation :

- L’indexation par type : cette approche offre une description formelle du document en utilisant ses métadonnées (type, auteur, titre, source, date, etc.), dont le vocabulaire est standardisé afin de faciliter l’utilisation de ces métadonnées par le plus grand nombre d’outils de recherche. Des documents semi-structures de type XML s’adaptent très bien à ce genre d’indexation. Dans ce cas, la structure (balises et attributs), peut jouer le rôle de métadonnées : on parle alors d’indexation structurelle de documents.
- L’indexation par concepts ou mots-clés : cette approche se focalise plutôt le contenu du document pour faciliter les opérations de recherche. Il peut s’agir, pour le concepteur du système, de recenser les termes qui apparaissent le plus souvent ; on parle alors d’indexation statistique. Il peut aussi s’agir d’un système plus évolué où le concepteur sélectionne les termes dans une base terminologique en rapport avec le document.

On peut parler de deux genres ou de deux niveaux d’indexation structurelle [2]. D’une part, on peut utiliser la composante structurelle pour effectuer une indexation structurelle dans le but d’établir leur classification structurelle. D’autre part, elle peut aussi donner un index de second niveau dans le but d’effectuer une interrogation efficace de ces documents. Dans notre travail, nous nous intéresserons plutôt à l’indexation permettant d’établir une classification structurelle de ces documents, mais nous ferons un tour rapide sur les méthodes d’indexation orientées interrogation.

2.2. Approches de l'indexation de l'information structurelle de documents XML

Nous allons présenter les approches de l'indexation de l'information structurelle de documents XML en se focalisant surtout sur celles utilisées dans le cadre de la classification structurelle ou de l'interrogation de schémas XML. Mais avant, nous allons donner quelques brefs exemples d'approches d'indexation structurelle en vue d'une interrogation. Tout d'abord, il convient d'introduire et de donner des définitions utiles de certains concepts concernant les arborescences.

2.2.1. Définitions préliminaires

Définition 2.1 (Arbre) Un arbre est un graphe orienté, connexe sans cycle tel que :
– Il existe un nœud et un seul où il n'arrive aucun arc. Ce nœud s'appelle la racine.
– Il arrive un arc et un seul en tout autre nœud.

Définition 2.2 (Chemin) Un chemin dans l'arbre est une liste $C = (x1, x2, ..., xk)$ de nœuds telle qu'il existe un arc entre chaque paire de nœuds successifs $i = 1, ..., k - 1$ $(xi, xi+1)$ à l'ensemble des arcs. La longueur du chemin correspond au nombre d'arcs parcourus, c'est-à-dire $k - 1$.

Définition 2.3 (Arbre ordonné) Un arbre est dit ordonné si les fils de chacun de ses nœuds (non feuilles), sont liés par une relation d'ordre totale de gauche à droite.

Définition 2.4 (Arbre étiqueté) Soit E un ensemble d'étiquettes. Un arbre T étiqueté par E est défini par (T, φ) tel que φ est une application qui associe à chaque nœud de T une étiquette de E.

Définition 2.5 (Sous-arbre fréquent) Un sous-arbre fréquent est un sous-arbre apparaissant dans la plupart des arbres d'une forêt considérée.

Définition 2.6 (Inclusion) Il existe de nombreuses manières de définir l'inclusion entre deux arbres $T1$ et $T2$. Toutes ces définitions se basent sur l'existence d'une application de mapping (correspondance) des nœuds de $T1$ vers les nœuds de $T2$. Ce mapping préserve de manière faible ou forte les étiquettes des nœuds et la structure de l'arbre. Les différences proviennent

des propriétés de préservation qui sont considérées, ainsi que de l'injectivité ou non du mapping [2].

Dans la Figure 2.1 est donné l'exemple d'un arbre *T*1 inclus dans *T*2 selon l'inclusion stricte exacte ordonnée. Mais les arbres *T*3 à *T*7 ne sont pas inclus dans *T*2 selon cette définition.

Cependant, il existe une relation d'inclusion stricte exacte entre *T*3 et *T*2 (Figure 2.2), et une relation d'inclusion stricte ordonnée entre *T*4 et *T*2 (Figure 2.3). Entre *T*5 et *T*2, il y'a une relation d'inclusion stricte (Figure 2.4). Et entre *T*6 et *T*2, on peut trouver une relation d'inclusion stricte faible (Figure 2.5). Enfin, entre *T*7 et *T*2, il existe une relation d'inclusion exacte ordonnée (Figure 2.6).

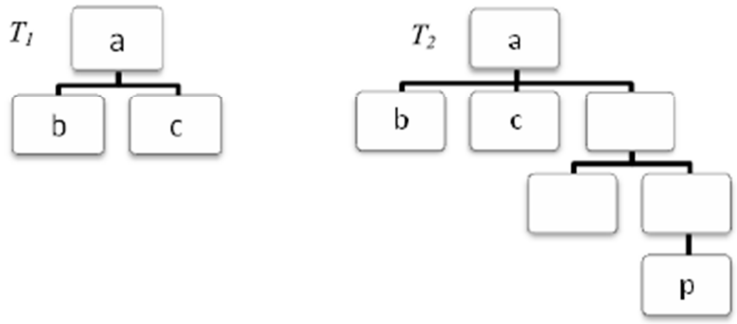


Figure 2.1 – Inclusion stricte exacte ordonnée

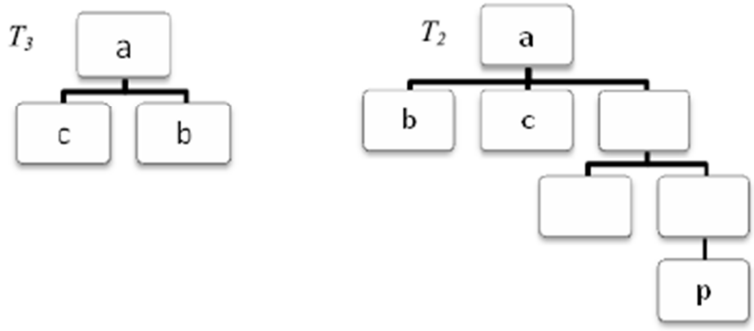


Figure 2.2 – Inclusion stricte exacte non ordonnée

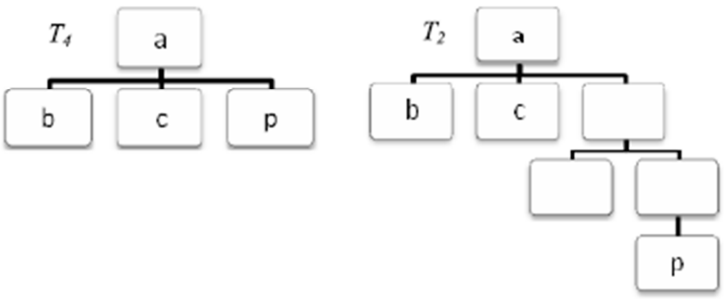


Figure 2.3 – Inclusion stricte ordonnée non exacte

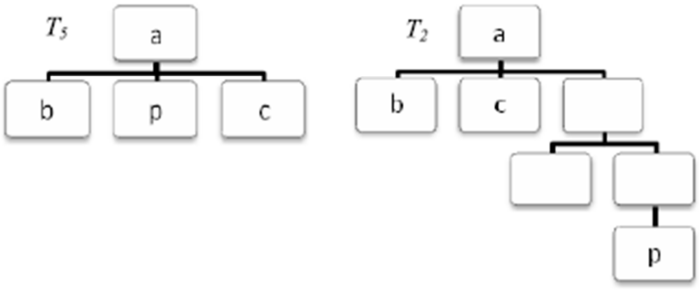


Figure 2.4 – Inclusion stricte non ordonnée non exacte

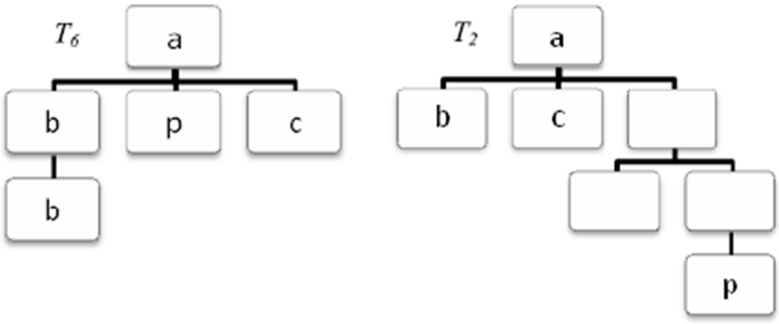


Figure 2.5 – Inclusion stricte faible non ordonnée, non exacte

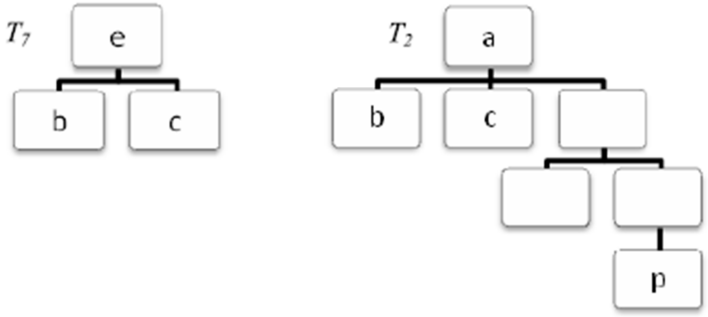


Figure 2.6 – Inclusion non stricte, mais exacte et ordonnée

Les documents informatiques structurés se prêtent particulièrement bien à la mise sous forme arborescente, comme c'est notamment le cas de documents XML, qui ont été conçus afin de stocker l'information sous une forme structurée [2]. La Figure 2.7 montre un exemple de document XML et la structure arborescente associée.

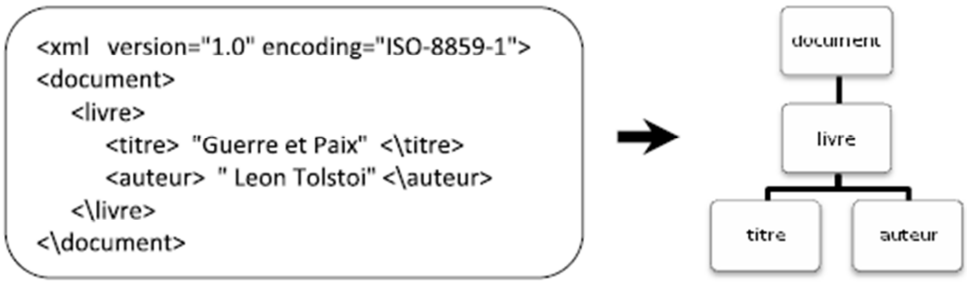


Figure 2.7 – Un document XML simple et l'arbre des balises associé

Notons que cette structure correspond à un arbre étiqueté ordonné. En effet, chaque nœud a comme étiquette une balise du document XML. En outre, l'ordre séquentiel des balises dans le document XML se traduit par l'ordre des nœuds-fils d'un nœud donné dans l'arbre correspondant. Il est donc possible de considérer cette structure arborescente comme un index de l'information structurelle du document XML source. Notons cependant que ce n'est pas toute l'information structurelle qui est utilisée pour former un index.

Très souvent, c'est une forme générique très représentative où l'on élimine toute forme de redondance en évitant autant que possible la perte d'information [2].

2.2.2. Approches d'indexation de l'information structurelle en vue d'une interrogation

Selon l'auteur [9], il existe plusieurs approches d'indexation de l'information structurelle dans le but d'une interrogation, parmi celles-ci on a l'indexation basée sur des champs et l'indexation basée sur des chemins que nous allons détailler de manière brève ici.

2.2.2.1. Indexation basée sur des champs

Dans cette approche, un document est représenté comme un ensemble de champs et de contenu associé à ces champs. Afin de procéder à une recherche concise limitée à certains

champs, les termes de l'index sont construits en combinant le nom d'un champ avec les termes du contenu [9]. Ceci est illustré dans la Figure 2.8 suivante.

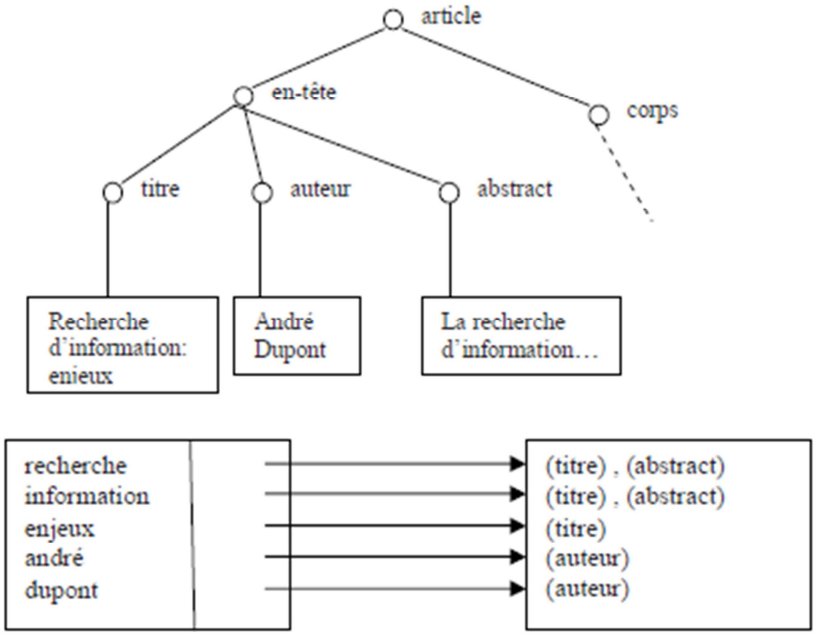


Figure 2.8 – Exemple d'indexation basée sur des champs

Les différents champs d'un document peuvent être obtenus de plusieurs façons :

- Ils peuvent être codés en tant que métadonnées dans les fichiers XML.
- Dans le cas d'un document d'un format quelconque transformé en XML, ils peuvent provenir du document dans son format original.

Ils peuvent être retrouvés en utilisant différentes techniques d'extraction [4].

- Ils sont extraits de la DTD ou du Schéma XML associé.

2.2.2.2. Indexation basée sur des chemins

La finalité des techniques basées sur des chemins est de retrouver efficacement et rapidement des documents ayant des valeurs connues pour des éléments ou attributs. Il s'agit également de simplifier la navigation pour résoudre efficacement des expressions d'interrogation de type Xpath et utiliser des index pleins textes sur les contenus.

Il résulte de ceci que les solutions proposées utilisent des index de chemins, c'est-à-dire des index donnant pour chaque valeur répertoriée d'un chemin de balises de type Xpath la liste des documents répondant contenant un élément atteignable par ce chemin et ayant cette valeur. La Figure 2.9 suivante illustre une indexation basée sur des chemins.

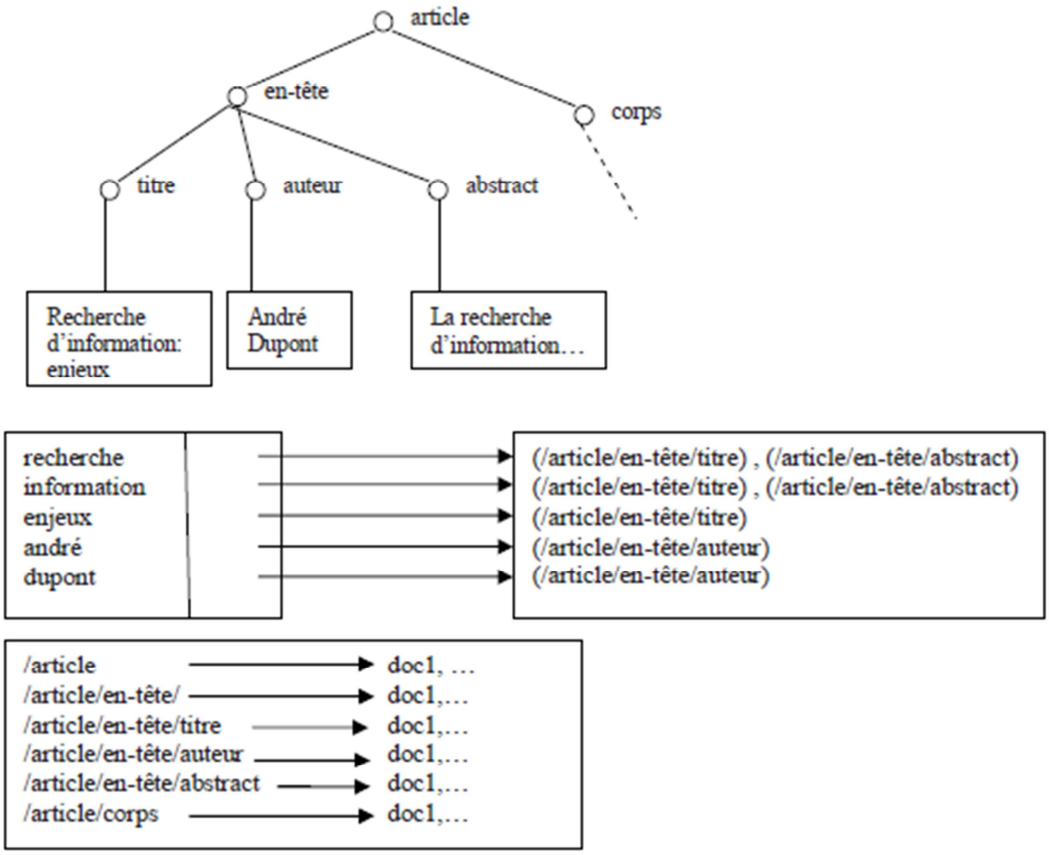


Figure 2.9 – Exemple d'indexation basée sur des chemins

Les deux approches précédentes ont en commun la difficulté de retrouver les relations ancêtres-descendants entre les différents nœuds des documents.

2.2.3. Approches d'indexation de l'information structurelle en vue d'une classification

Nous allons détailler ici les différentes méthodes d'indexation en vue d'une classification selon les différents auteurs de la littérature.

2.2.3.1. Indexation basée sur les sous-arbres fréquents

A l'origine, la recherche de sous-arbres fréquents dans des ensembles d'arbres a été motivée par l'analyse de documents électroniques. Plusieurs approches ont été tentées [6], certaines parmi celles-ci utilisent comme données d'entrée des traces de navigation dans des sites web, ou encore des documents informatiques mis sous forme d'arbres comme les documents XML.

Etant donné une collection d'arbres XML, c'est-à-dire une base de structures arborescentes associées à un ensemble de documents XML, un sous-arbre fréquent se retrouve dans la majorité des structures arborescentes XML de cette collection, mais on peut retrouver un ou plusieurs sous-arbres fréquents dans un même arbre XML. C'est-à-dire qu'un sous-arbre fréquent peut structurellement indexer plusieurs documents XML (intégration), il est également possible de voir un même document XML indexé par plusieurs sous-arbres fréquents (recouvrement). Plusieurs sous-arbres fréquents peuvent représenter l'index de l'information structurelle d'un document XML. Cependant, les sous-arbres fréquents peuvent être distingués selon qu'ils soient ordonnés ou non et/ou exacts ou non. Il est nécessaire de traiter l'ordre si celui-ci est spécifié dans le cas de l'intégration de DTDs ou de schémas XML [2].

Les différentes approches diffèrent sur la définition de l'inclusion d'un arbre dans un autre. Pour certaines, les sous arbres fréquents apparaissent selon une inclusion stricte et exacte (induite), et nécessairement ordonnée, comme c'est le cas de la Figure 2.1 et la Figure 2.10. L'ordre peut être indirect comme dans la Figure 2.11. En revanche, d'autres approches sont plus flexibles, car elles considèrent qu'un arbre est inclus dans un autre comme c'est le cas pour l'arbre T_4 dans la Figure 2.3 et pour l'inclusion entre les arbres T_1 et T_2 dans la figure 2.12, même si l'imbrication de ces nœuds diffère de celle de l'inclusion stricte et exacte.

Cela permet de distinguer des sous-arbres fréquents même si les structures des arbres XML de la collection considérée sont hétérogènes. Cependant certaines approches s'alignent sur le fait que les descendants d'un nœud doivent toujours avoir le même ordre pour composer un sous-arbre fréquent. L'approche [1] généralise les approches précédentes et offre moins de

contraintes dans l'inclusion. Autrement-dit, elle considère l'inclusion d'un arbre dans un autre sans tenir compte de l'ordre des frères comme dans les Figure 2.4 et 2.13.

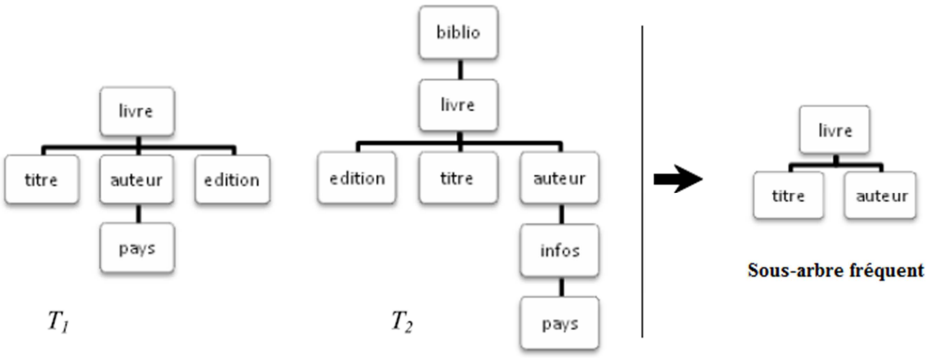


Figure 2.10 – Arbre fréquent selon l'inclusion stricte exacte ordonnée

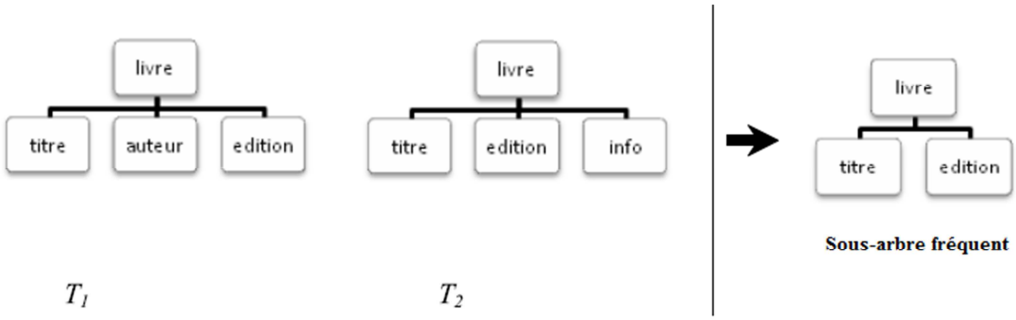


Figure 2.11 – Arbre fréquent selon l'inclusion stricte exacte et ordonnée indirecte

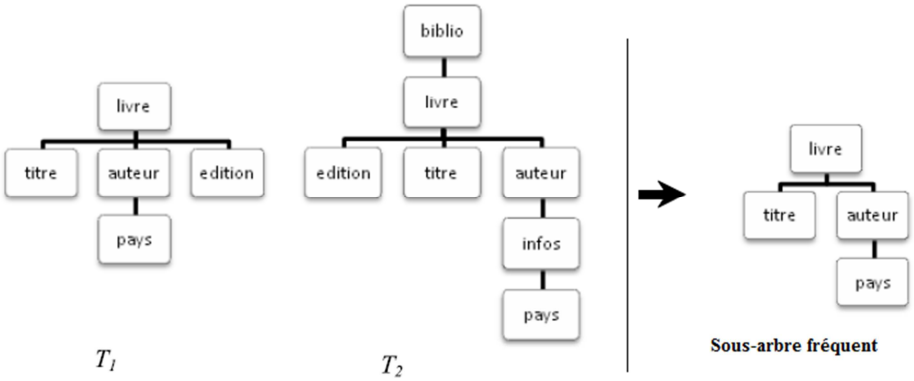


Figure 2.12 – Arbre fréquent selon l'inclusion stricte ordonnée non exacte

L'indexation de l'information structurelle de documents XML basée sur le formalisme des sous-arbres fréquents considère que plus il y a des sous-arbres fréquents communs entre deux documents XML, plus ils sont proches structurellement. On notera un gros inconvénient

qui est que la différence de taille entre les structures arborescentes n'affecte en aucun cas les propriétés d'inclusion. Autrement dit, si la relation d'inclusion au sens sous-arbre fréquent est satisfaite entre un ensemble de documents XML, même de longueurs différentes, ceux-ci seront accessibles via un même index [2].

La Figure 2.13 schématise ce cas. Effectivement, T_1 et T_2 sont des sous-arbres fréquents distincts inclus dans l'arbre T_3. Cependant, le fait qu'ils partagent la structure T_3 semble supposer qu'ils sont similaires à T_3 indépendamment l'un de l'autre. Donc les documents XML correspondant à T_1 et T_3 seront regroupés dans une même classe (de même pour T_2 et T_3) car ils seront retournés par le système pour une même requête même s'ils contiennent des informations non significatives par rapport à la requête formulée. Un autre inconvénient majeur est la complexité engendrée par les algorithmes d'extraction des sous-arbres fréquents [2].

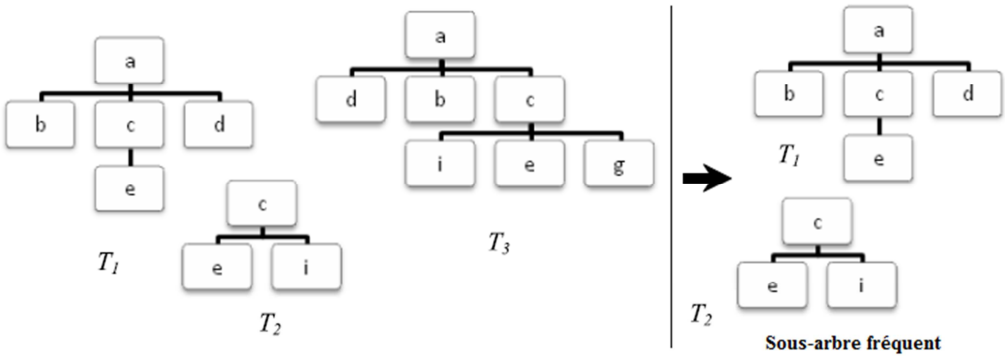


Figure 2.13 – Extraction d'arbres fréquents selon la subsumption

2.2.3.2. Indexation basée sur les vecteurs structurés

Le modèle de représentation d'un document XML par un vecteur structuré SVM (Structured Vector Model) a été proposé par [5]. C'est un vecteur composé de termes (pour le texte), ou un autre vecteur structuré (récursivité), qui représente l'arborescence de la structure du document. A l'origine, il a été conçu pour tenir compte de la structure et du contenu simultanément, mais il peut être aisément détourné vers la structure uniquement.

La Figure 2.14 et la Figure 2.15 illustrent dans l'ordre un modèle d'arbre XML et le vecteur SVM associé. Mais afin d'intégrer plus facilement le formalisme du SVM, il faut tout d'abord définir certaines notions ainsi que leurs significations respectives (Table 2.1).

Notation	Signification
$e_d(0,0)$	Racine du document *d*
$e_d(i,j)$	j^{eme} nœud du niveau *i*
$m_d(i,j)$	Nombre de fils du nœud $e_d(i,j)$
$m_d(i)$	Nombre de nœuds du niveau *i* de *d*
h_d	Hauteur de la structure arborescente de *d*
$t_d(i,j)$	Texte enchâssé dans $e_d(i,j)$
$\overrightarrow{t_d(i,j)}$	Vecteur représentant le nœud feuille$t_d(i,j)$
$\overrightarrow{e_d(i,j)}$	Vecteur représentant le nœud $e_d(i,j)$

Table 2.1 – Notations utilisées pour le SVM

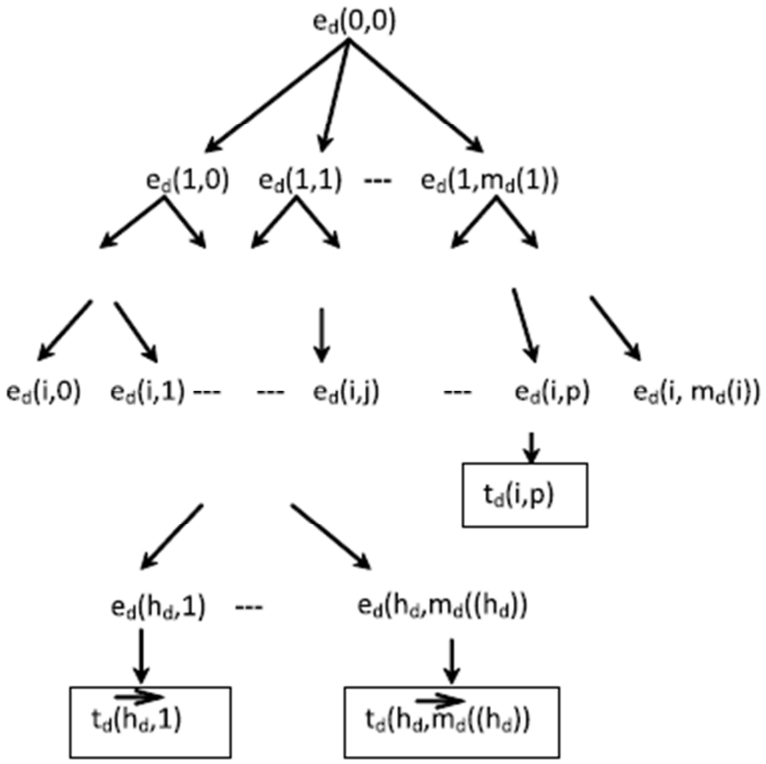


Figure 2.14 – Arbre XML basé sur les notations SVM

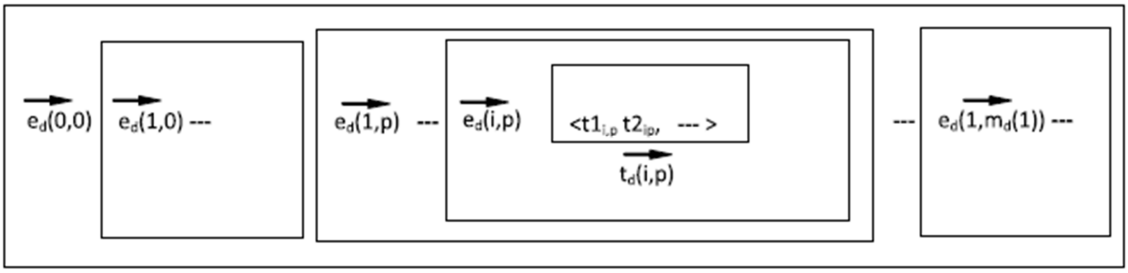


Figure 2.15 – Modèle SVM d'un arbre XML

Pour effectuer une comparaison structurelle de deux documents XML, le modèle SVM est exploité en comparant leurs chemins respectifs. Des expériences ont été menées sur deux corpus XML de petite taille possédant des structures simples, car les modèles ont des difficultés à s'adapter à des structures trop complexes.

Il existe une extension du modèle [5] qui prend en compte les liens entre les documents dans un SVM (s'ils existent). Cette extension est la représentation SLVM (Structured Link Vector Model), où les termes, la structure et le voisinage des documents composent le vecteur. Il faut noter que tenir compte des liens va donner lieu à un graphe et non plus à une arborescence simple.

2.2.3.3. Indexation basée sur les chemins (paths)

La représentation (indexation) structurelle des documents XML est souvent effectuée par des arbres étiquetés ; ces étiquettes correspondent aux balises, et éventuellement aux noms des attributs. Un chemin est le plus court trajet entre la racine d'un arbre et un nœud considéré. Un sous-chemin est un segment du chemin.

Dans certaines approches, des documents XML ont été représentés par différents sous-ensembles de chemins ou sous-chemins. C'est une linéarisation des arbres XML permettant de comprimer leur arborescence en conservant leurs structures, donc de rendre les traitements moins complexes. Il est préférable de prendre en compte les sous-chemins et pas seulement les chemins car la similarité de structure entres les documents peut se trouver à des niveaux variant. Pour des ensembles de documents XML qui serait très hétérogènes, les ressemblances et différences feront leur apparition dès les premiers niveaux (pas loin de la racine).

Cependant pour des ensembles de documents plus homogènes, il sera plus intéressant de ne garder que les sous-chemins qui seront proches des feuilles.

Chaque chemin est considéré comme un mot. Une dépendance entre ces derniers peut poser un problème à cause de ce choix. Ceci est d'une grande importance quand on travaille avec des algorithmes qui considèrent que les mots sont indépendants. Pour contrer cet inconvénient, ces chemins seront distingués selon leurs longueurs. Dans la Table 2.2 on présente un ensemble de chemins avec leurs fréquences respectives pour l'exemple de la Figure 2.16.

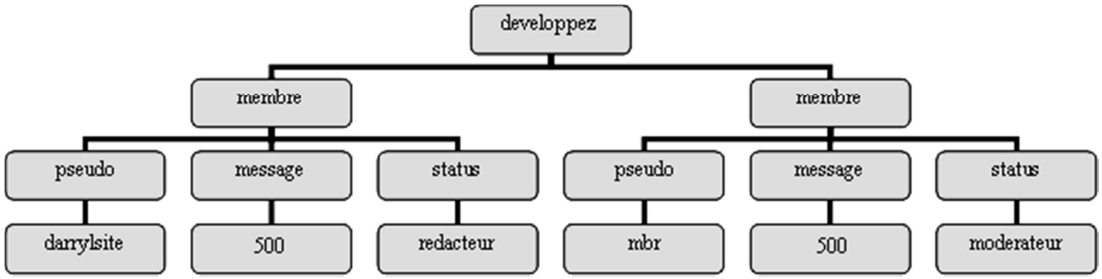


Figure 2.16 – Arbre XML avec son contenu

Mots	Fréquence
developpez.membre.pseudo.darryliste	1
developpez.membre.message.500	2
Developpez.membre.status.moderateur	1

Table 2.2 – Paths de longueur 4 avec leurs fréquences respectives

2.2.3.4. Indexation basée sur les niveaux

L'indexation basée sur les niveaux est une approche utilisée pour classifier structurellement des documents XML en parcourant leurs structures arborescentes niveau par niveau (Depth First Taversal). Tout comme dans la majorité des approches, les documents XML sont représentés par des arbres étiquetés. Cependant, au lieu de les parcourir en profondeur, on les parcourt par niveaux. Ainsi, les niveaux des nœuds visités représentent des portions d'index de l'information structurelle d'un document XML [2].

On pourra donc citer la possibilité de comprimer (aplatir) ces structures arborescentes (tout comme l'approche d'indexation par les paths), et les transformer en plusieurs sous-ensembles que l'on nommera niveaux. En pratique, il est souhaitable de calculer la similarité progressivement par niveaux que de pratiquer des calculs directs entre deux arbres.

Dans une subdivision, chaque niveau peut être exploité individuellement en comparant deux arbres ; afin de connaitre le contexte hiérarchique des niveaux, c'est-à-dire les niveaux ancêtres et les niveaux descendants, il est nécessaire de connaitre leurs positions dans leurs arbres respectifs ; ceci permettra d'offrir une certaine fiabilité dans le calcul des similarités de leurs arborescences respectives.

La Figure 2.17 suivante montre un exemple d'indexation basée sur les niveaux d'un arbre.

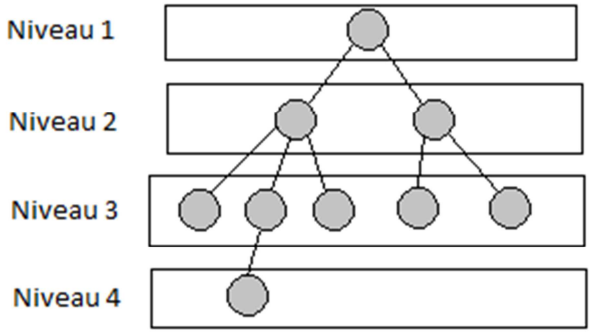


Figure 2.17 – Indexation basée sur les niveaux d'un arbre

2.2.3.5. Indexation basée sur les bitmaps

Les auteurs [7] ont proposé deux façons de représenter un document. La première utilise un index de bitmap qui est une matrice à deux dimensions : les documents et les chemins des feuilles à partir de la racine. La Figure 2.18 illustre deux documents XML simples, et la Table 2.3 représente l'index bitmap associé.

Soient l'ensemble de chemins :

- P0=A.B.C ;
- P1=A.B.C.D.E ;
- P2=A.H.

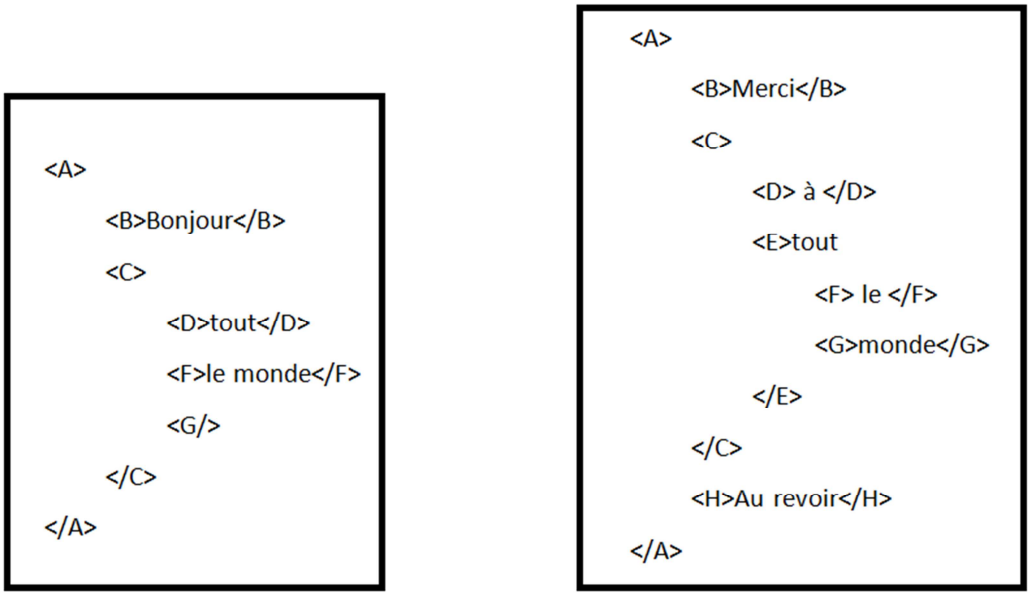


Figure 2.18 – Exemple de documents XML

L'index bitmap se construit de telle sorte que si un chemin Pi appartient à un document dj, le bit de celui est mis à 1, sinon il sera mis à 0 ; comme illustré dans la Table 2.3.

L'aspect essentiel qui caractérise cette approche est que la représentation est directement obtenue à partir du document XML source et des divers paths considérés, contrairement à « l'indexation basée sur les paths » et « l'indexation basée sur les niveaux » dans lesquelles la représentation est issue de l'arborescence du document XML.

	p0	p1	p2
d1	1	0	0
d2	1	1	1

Table 2.3 – Un index bitmap des documents XML de la figure 2.16

La première représentation a été étendue à 3 dimensions pour tenir compte en plus du contenu des documents. Le bitmap à 3 dimensions (BitCube) d'un document XML est représenté par le quadruplet (*d*,*p*,*v*,*b*), où *d* est le document XML, *p* est un path, *v* un terme

(ou un contenu) d'un chemin et *b* un booléen. Celui-ci est positionné à *vrai* si *p* contient le terme *v*, sinon il sera positionné à *faux*. Ce type de représentation permet d'améliorer et d'optimiser les requêtes d'interrogation de l'ensemble de documents.

2.2.3.6 Indexation basée sur les arbres et résumés d'arbres

L'approche la plus simple d'indexation structurelle d'un document XML est de donner sa représentation arborescente. Cependant, certaines approches utilisent des formes plus générales à la place de cette représentation pour des raisons d'optimisation. Ces structures de remplacement peuvent être soit un arbre étiqueté qui correspond à la structure d'origine du document XML, soit un « résumé d'arbre ».

Les différents auteurs divergent sur les méthodes de réalisation de résumés d'arbres. Selon les auteurs [10], un résumé d'arbre est obtenu par deux transformations : la première pour éliminer les nœuds imbriqués et répétés, et la seconde pour éliminer les duplications. L'imbrication et la duplication des éléments est la principale raison qui fait que des documents XML, même issus d'une source de données utilisant une même DTD, peuvent différer structurellement. Un nœud imbriqué-répété est un nœud non-feuille qui renfermerait la même information que l'un de ses ancêtres. Selon un parcours pré-fixé d'un arbre, un nœud dupliqué est un nœud dont le path (chemin allant de la racine jusqu'au nœud considéré) a déjà été parcouru. La Figure 2.19 a un exemple de redondance : l'arbre T1 et T3 diffèrent à cause des nœuds A (imbriqué – répété) et B (répété). La réduction d'imbrication et la réduction de répétition seront utilisées pour extraire la structure arborescente résumée des arbres étiquetés ordonnées et enracinés qui représentent les documents XML. Chaque type de réduction préconise seulement un parcours pré-fixé sur l'arbre original du document.

La réduction d'imbrication va éliminer les imbrications dans l'arbre original, ce qui aura pour conséquence l'élimination également des nœuds imbriqués-répétés. On effectuera un parcours pré-fixé de l'arbre dans le but de détecter les nœuds qui auraient le même label que l'un de leurs ancêtres, et cela pour déplacer leurs nœuds fils vers le haut (les mettre au même niveau). Cependant, ce processus peut causer répétition de certains nœuds. C'est pour cela qu'il est préférable de s'intéresser au préalable à la réduction d'imbrication, puis de passer à la réduction de répétition. La réduction de répétition élimine les répétitions de nœuds dans l'arbre original du document. Dans ce cas, on utilisera également un parcours pré-fixé de l'arbre en ignorant les paths déjà parcourus et en ne gardant que les nouveaux, et cela en utilisant une table de hashage.

Dans le figure 2.19 est présenté un exemple d'extraction de structure arborescente résumée à partir de T1, en appliquant l'étape de réduction d'imbrication sur ce dernier, ce qui donnera l'arbre T2, où il n'y a plus de nœuds imbriqués-répétés. Puis en appliquant la réduction de répétition sur T2, on obtiendra T3 qui est l'arbre résumé sans nœuds imbriqués-répétés ou nœuds répétés.

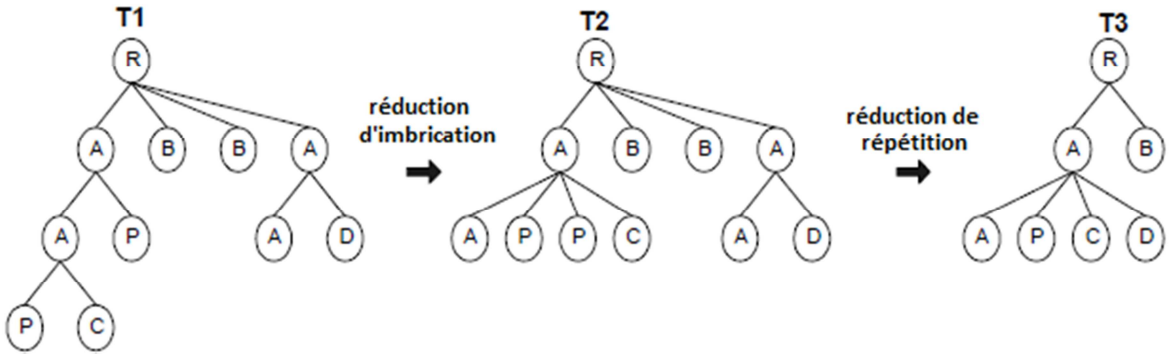


Figure 2.19 – Extraction de résumés d'arbres selon [10]

Les auteurs dans [2] représentent également un document XML par un résumé d'arbre étiqueté ordonné et enraciné, à la différence que ce dernier est obtenu en éliminant seulement les nœuds frères dupliqués et en transformant les attributs éventuels en descendants directs (fils) du nœud (balise) auquel ces attributs sont rattachés dans le document XML. La Figure 2.20 illustre cette méthode de représentation et montre toutes ses caractéristuques.

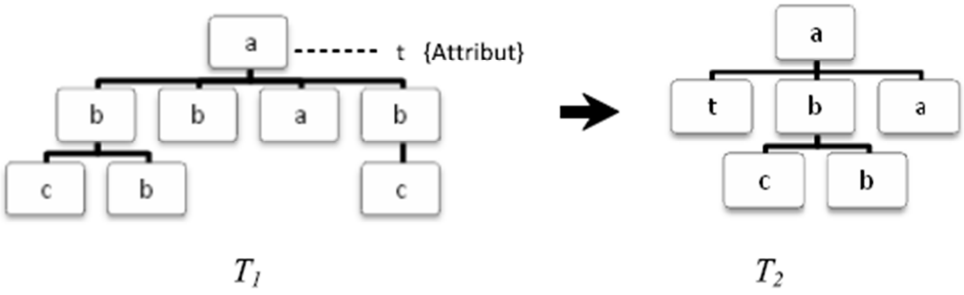


Figure 2.20 – Extraction de résumé d'arbre selon [2]

Effectivement, en transformant l'arbre original T1 en le résumé d'arbre T2 montre que l'attribut « t » devient un nœud fils du nœud « a ». On remarque aussi que la duplication des nœuds frères « b » est éliminée tout en gardant la descendance directe (« c », « b » et « c ») qui sera rattachée à une seule occurrence de « b », mais les nœuds « c » qui étaient à l'origine

cousins sur T1 sont devenus frères, dont il a fallu éliminer la duplication dans T2. Mais comme cité précédemment, les duplications des nœuds imbriqués (« a » et « b ») sont maintenues.

Dans cette méthode d'extraction de résumés d'arbres, la hiérarchie de la structure est bien conservée, contrairement à la méthode précédente [10] où celle-ci pourrait totalement changer, en effet, la hauteur de l'arborescence dans la Figure 2.19 est passée de 4 à 3 niveaux à cause particulièrement de l'étape de réduction d'imbrication qui élimine toute répétition de nœud partant du nœud racine aux feuilles sur tous les chemins. C'est pour cela, dans notre travail de programmation nous nous sommes focalisés sur la seconde méthode [2] qui impose moins de contraintes et offre plus d'avantages comme un traitement plus rapide et aisé des arbres et aucune perte d'information, nous verrons cela dans le chapitre suivant.

Conclusion

Dans ce chapitre nous nous sommes intéressés à l'indexation et aux différentes méthodes d'indexation de l'information structurelle de documents XML en vue d'une classification structurelle, et particulièrement à la méthode qui nous intéresse dans le cadre de notre travail qui est la méthode d'indexation basée sur les résumés d'arbres.

Chapitre III

Approche d'indexation de documents XML par des résumés d'arbres

Introduction

Nous allons nous étaler sur l'approche d'indexation de documents XML en se basant sur le formalisme des arbres résumés, et particulièrement l'approche des auteurs [2].

3.1. Extraction du résumé d'arbre structurel

Dans ces approches, on se propose de représenter les documents XML par leurs structures de résumés d'arbre. Un résumé d'arbre est une forme arborescente générique représentant un document XML, dans laquelle on n'aurait gardé que l'essentiel de l'information de ce dernier, c'est-à-dire en excluant toute redondance inutile.

Comme cité dans le chapitre II précédent, les approches d'extraction d'arbres résumés diffèrent d'un auteur à un autre. Selon l'auteur [10], il faut deux étapes ou transformations pour extraire le résumé d'arbre d'un document XML : la première a pour but d'éliminer les imbrications de nœuds, c'est-à-dire les nœuds qui se répètent sur un chemin partant de la racine à ce nœud (nœud non feuille) ; la seconde a pour rôle d'éliminer les duplications de nœuds c'est-à-dire les nœuds du même niveau (frères) qui se répètent. Cette méthode présente un gros risque de perte d'information structurelle car l'élimination des imbrications des nœuds modifie la structure arborescente en réduisant la profondeur de l'arbre.

Dans la Figure 3.1, on a un exemple d'extraction de structure d'arbre résumé à partir de l'arbre T1 contenant des nœuds imbriqués et répétés. Avec l'application de la première

étape, c'est-à-dire l'élimination des imbrications, on remarque clairement que la profondeur de l'arbre est passée de 4 niveaux à 3 niveaux, ce qui représente une perte d'information structurelle. Après la seconde étape on remarque l'élimination des duplications des nœuds 'B' (2e niveau) et 'P' (3e niveau).

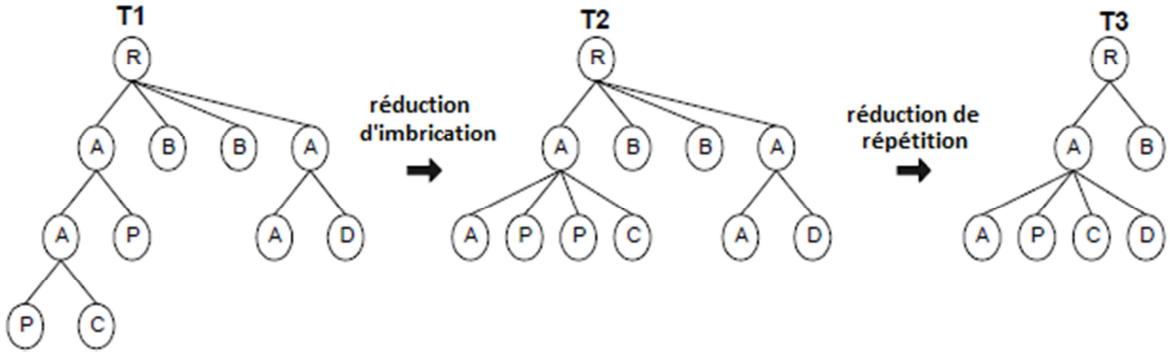


Figure 3.1 – Extraction de résumé d'arbre en deux étapes selon [10]

L'approche des auteurs [2] représente également un document XML par sa structure générique ou résumé d'arbre. Cependant, dans ce cas l'extraction de ce résumé d'arbre diffère de celle de l'approche précédente. La différence majeure est que dans cette approche, il n'est procédé qu'à l'élimination des nœuds du même niveau hiérarchique (frères) qui se répètent, et aussi tout tag optionnel (attribut) est récupéré en tant que descendant direct du nœud (balise) auquel il est rattaché dans le document XML d'origine. Et comme cité précédemment, cette approche est plus appropriée que celle de [10] car elle évite la perte d'information en récupérant les attributs et en gardant la même structure hiérarchique que l'arbre du document XML d'origine, c'est la raison pour laquelle c'est cette approche que nous avons choisie pour réaliser notre application, en suivant les travaux de [2].

Les auteurs [2] ont réalisé cette extraction de résumé d'arbre à partir d'un document XML en suivant un algorithme qu'ils ont créé. La démarche à suivre est en deux étapes. La première étape s'appuie sur SAX, qui renvoie tous les tags et attributs rencontrés sur un document XML. Ces derniers sont interceptés, filtrés puis transformés en une forme intermédiaire parenthèsée. Cette forme intermédiaire est composée seulement de parenthèses

ouvrantes et fermantes ainsi que de tags et/ou attributs. La profondeur des parenthèses définit les niveaux hiérarchiques des éléments XML. Ainsi deux parenthèses à l'intérieur de deux autres parenthèses signifient que le premier élément est un descendent du second. A chaque fois que le parseur SAX rencontrera dans le document XML qu'il analyse une balise ouvrante ou fermante, il déclenchera l'événement approprié en faisant appel à la méthode correspondante. Ainsi une balise ouvrante donnera lieu à une parenthèse ouvrante, et inversement, une balise fermante à une parenthèse fermante. SAX se compose de plusieurs méthodes, qui chacune est déclenchée à la rencontre d'un événement particulier. Nous verrons chacune de ces méthodes dans le prochain chapitre. Le choix d'utiliser cette forme parenthèsée a été motivé par la simplicité de cette représentation. Effectivement, il est beaucoup plus simple de lire la forme arborescente d'un document XML sous cette forme que sous sa forme originale, car cette forme est plus concise. Cette représentation parenthèsée est aussi dénuée de tout contenu, ce qui la rend légère et facilite la distinction des différentes balises et hiérarchies d'un document XML. L'autre avantage de cette forme est qu'elle facilite grandement l'étape suivante qui consiste en l'extraction de la forme d'arbre résumé.

La seconde étape est l'étape d'extraction de l'arbre résumé à partir de la forme parenthèsée qui résulte de l'étape précédente. Au cours de Cette phase, la forme intermédiaire, produite par la première phase, est transformée par un autre parseur en le résumé d'arbre correspondant conformément aux prévisions de l'approche, c'est-à-dire en éliminant les duplications des nœuds frères, et en considérant chaque attribut comme un descendant direct (fils) de l'élément (tag) auquel il est rattaché dans le document XML source. L'essentiel de la tâche d'extraction est accompli par ce deuxième parseur, car c'est lui qui permet de passer de la forme linéaire (source) du document à sa représentation hiérarchique (résumé d'arbre). En somme, trois opérations essentielles sont effectuées à ce niveau :

1- Passage de la forme linéaire du document à sa représentation hiérarchique ;
2- Suppression des duplications des nœuds frères ;
3- Transformation des attributs éventuels en descendants directs des éléments auxquels ils sont rattachés dans le document XML source.

Ce parseur agit sur la forme parenthèsée extraite de la phase précédente en utilisant le principe de la précédence des opérateurs, c'est-à-dire entre les symboles de parenthèse ouvrante et

parenthèse fermante. Ce principe est très utile dans le cas d'analyse syntaxique de phrases, donc il facilitera grandement l'extraction de l'arbre résumé.

L'algorithme d'extraction de résumés d'arbres des auteurs [2] est schématisé dans la Figure 3.2.

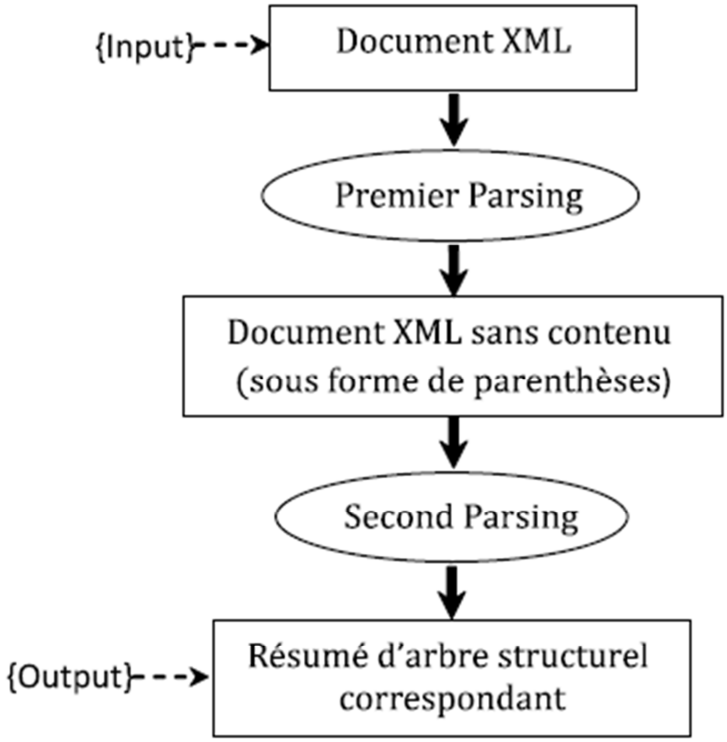


Figure 3.2 – Algorithme d'extraction de résumés d'arbres selon [2]

Nous proposons maintenant un exemple concret sur lequel nous allons appliquer les différentes étapes de l'algorithme précédent, en mettant en avant les résultats de chaque étape. Dans la Figure 3.3 est schématisé le document livre.xml que nous avons utilisé comme exemple.

Comme on le remarque dans la figure, la balise mère est <book>. Cette balise a des balises filles <chapter> qui se répètent plusieurs fois. Chaque balise <chapter>, à son tour, a des balises filles comme <title> ou <subChapter> qui se répète plusieurs fois ; on peut aussi trouver des attributs nb_pages (optionnels) rattachés aux balises <chapter>.

Ce document XML contient un grand nombre de redondances, il fait donc un bon candidat à l'algorithme d'extraction de résumé d'arbre.

```
<book>
        <chapter>
                <title>Introduction</title>
        </chapter>
        <chapter nb_pages="27">
                <title>Récit 1</title>
                <subChapter>
                        <title>Partie 1</title>
                </subChapter>
                <subChapter>
                        <title>Partie 2</title>
                </subChapter>
        </chapter>
        <chapter nb_pages="50">
                <title>Récit 2</title>
                <subChapter>
                        <title>Partie 1</title>
                </subChapter>
                <subChapter>
                        <title>Partie 2</title>
                </subChapter>
                <subChapter>
                        <title>Partie 3</title>
                </subChapter>
        </chapter>
        <chapter nb_pages="36">
                <title>Récit 3</title>
                <subChapter>
                        <title>Partie 1</title>
                </subChapter>
                <subChapter>
                        <title>Partie 2</title>
                </subChapter>
        </chapter>
        <chapter>
                <title>Conclusion</title>
        </chapter>
        <chapter>
                <title>Index</title>
        </chapter>
</book>
```

Figure 3.3 – Livre.xml

L'extraction de la structure d'arbre résumée de ce document XML commence par la première phase qui est l'analyse avec le premier parseur, soit le parseur SAX. Le rôle de cette étape, comme cité précédemment, est d'arriver à la forme intermédiaire parenthèsée du

document XML, dénuée de tout contenu textuel, et ne gardant que l'information structurelle hiérarchique sous une forme linéaire.
La construction de cette forme parenthèsée se fera par l'appel des méthodes de SAX à chaque rencontre d'un événement. Le début du parsing du document XML est le premier évènement, puis surviennent d'autres événements au cours de l'analyse du document. L'ouverture d'une balise donnera lieu à l'appel de la méthode qui écrira une parenthèse ouvrante '('. Ainsi de suite pour les autres événements qui peuvent survenir comme la fermeture d'une balise qui donnera lieu à une parenthèse fermante ')', et d'autres entités qui seront ignorées dans notre cas tels les contenus textuels et les blancs. Le dernier évènement rencontré est évidemment la fin du fichier XML, qui fera que ce dernier sera fermé en lecture. La rencontre d'attributs éventuels donnera également lieu à une parenthèse ouvrante dans la forme parenthèsée car ces derniers seront considérés comme étant des descendants directs de la balise à laquelle chacun d'eux sera rattaché.

En appliquant ce premier parsing sur le fichier de notre exemple, voici dans la Figure 3.4 qui suit la forme intermédiaire parenthèsée que l'on obtient. On remarque clairement, le côté hiérarchique de cette représentation, car chaque parenthèse ouvrante représente une nouvelle balise ouvrante dans le document XML, et les attributs présents ont été reliés en tant que descendants direct des nœuds auxquels ils sont reliés.

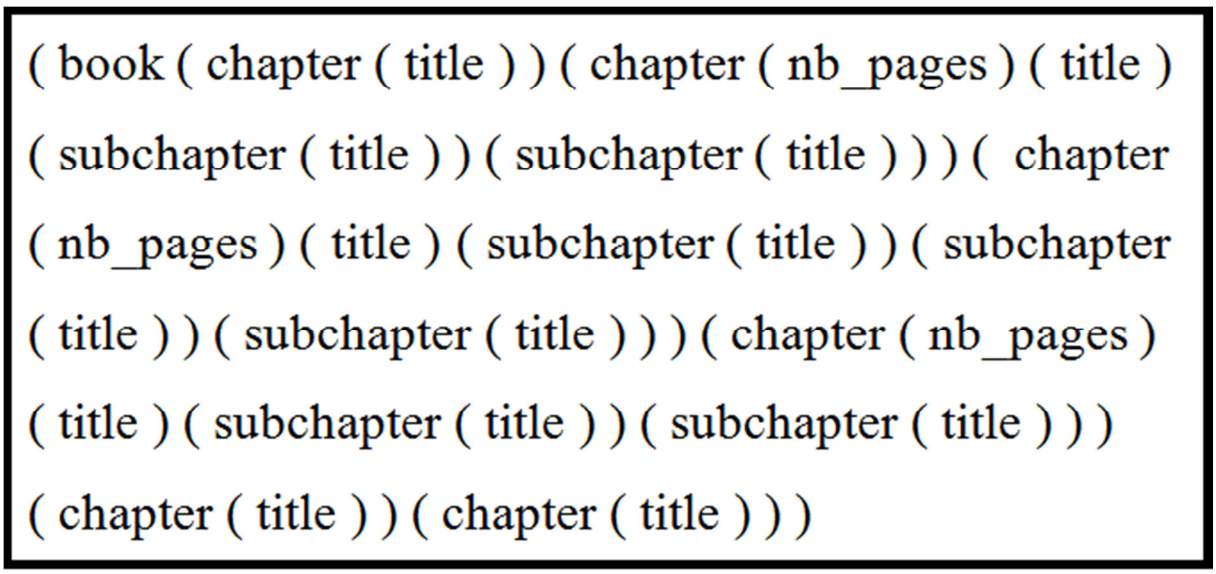


Figure 3.4 – Forme parenthèsée extraite de notre exemple

Une fois cette forme intermédiaire parenthèsée extraite, c'est le second parseur qui intervient sur cette dernière pour extraire la forme d'arbre résumée nœud par nœud. Comme cité précédemment, ce parseur se basera sur le principe de la précédence d'opérateurs pour analyser cette forme intermédiaire, symbole par symbole, dans le but de distinguer chaque nœud et sa descendance, il construira donc le résumé d'arbre progressivement en éliminant les duplications des nœuds du même niveau (frères) et en reliant les éventuels attributs aux nœuds auxquels ils sont reliés dans le document XML d'origine Livre.xml.

Le résumé d'arbre extrait de notre exemple est schématisé dans la Figure 3.5 suivante.

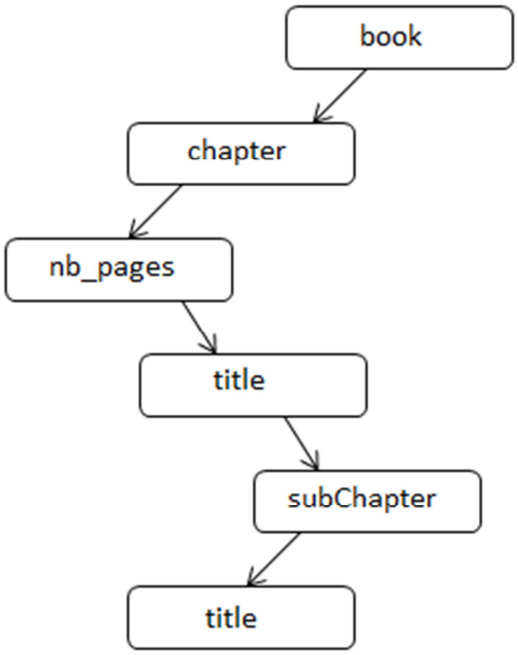


Figure 3.5 – Résumé d'arbre de Livre.xml

Cette structure arborescente résumée, comme son nom l'indique, résume le document XML qu'elle représente, soit Livre.xml dans notre cas. Cet arbre résumé se compose de six nœuds sur 4 niveaux. On remarque que cette forme de représentation est compacte et très simple à lire. Elle permet de connaitre le contenu essentiel du document XML sans redondance ni information inutiles ; en même temps elle garantit qu'aucune information structurelle indispensable à la cohérence du document XML n'a été perdue ou occultée.

Cette structure dans la Figure 3.5 représente un gros gain par rapport à l'utilisation de la structure arborescente complète du document XML qui est représentée dans la Figure 3.6 qui suit.

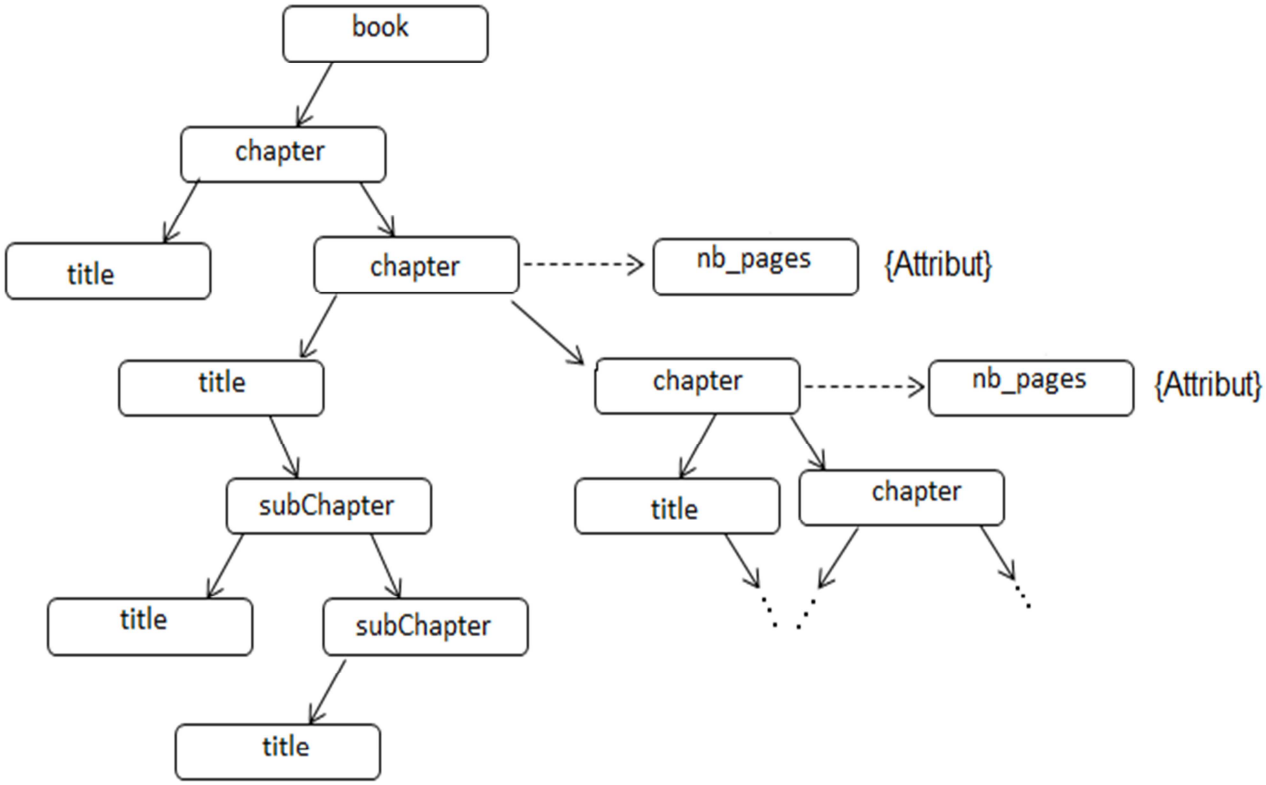


Figure 3.6 – Extrait de la structure arborescente complète (non résumée) de Livre.xml

On remarque, qu'il y a beaucoup plus de nœuds dans cette représentation que dans l'arbre résumé, ce qui résulterait de son utilisation une plus grande complexité de traitements informatiques, que ce soit pour son extraction ou pour son utilisation pour d'autres traitements.

Dans le cadre de notre travail, ce traitement d'extraction de résumés d'arbres se fait sur des collections de documents XML, c'est pour cela qu'il a été plus judicieux de notre part d'utiliser l'approche proposée par [2] qui nous offre plus d'avantages, et beaucoup moins de contraintes.

Conclusion

Nous avons vu dans ce chapitre les approches d’extraction de résumés d’arbre, et nous nous sommes penchés plus en détails sur l’approche des auteurs [2].

Chapitre IV

Implémentation et expérimentation de la méthode d'extraction des résumés d'arbres

Introduction

Le but de ce chapitre est de décrire et de détailler les méthodes et procédures utilisées pour réaliser notre application, et d'expliquer l'environnement de travail ainsi que les différentes interactions au sein de l'application, et nous allons donner au préalable quelques définitions de certains concepts utilisés.

4.1. Rappels et définitions

Définition 4.1 (pointeur)

Un pointeur est une variable contenant l'adresse d'une autre variable d'un type donné. La notion de pointeur fait souvent peur car il s'agit d'une technique de programmation très puissante, permettant de définir des structures dynamiques, c'est-à-dire qui évoluent au cours du temps, par opposition aux tableaux par exemple qui sont des structures de données statiques, dont la taille est figée à la définition.

Définition 4.2 (la précédence des opérateurs)

La priorité des opérateurs spécifie l'ordre dans lequel les valeurs doivent être analysées. Par exemple, dans l'expression *1 + 5 * 3*, le résultat est *16* et non *18*, car la

multiplication ("*") a une priorité supérieure par rapport à l'addition ("+"). Des parenthèses peuvent être utilisées pour forcer la priorité, si nécessaire. Par exemple : *(1 + 5) * 3* donnera *18*. Si la priorité d'opérateur est égale, l'associativité de gauche à droite est utilisée.

La table suivante représente quelques précédences d'opérateurs.

	(	)	*
(	<	=	
)	>	>	>
*	<		

Table 4.1 – Table de précédence de certains opérateurs

Définition 4.3 (pile)

La pile est une structure de données, qui permet de stocker les données dans l'ordre LIFO (Last In First Out). Pour l'implémentation j'ai choisi une liste simplement chaînée, présentée sur la verticale. L'insertion se faisant toujours au début de la liste, le 1er élément de la liste sera le dernier élément saisi, donc sa position est en haut de la pile.

La Figure 4.1 suivante illustre une pile.

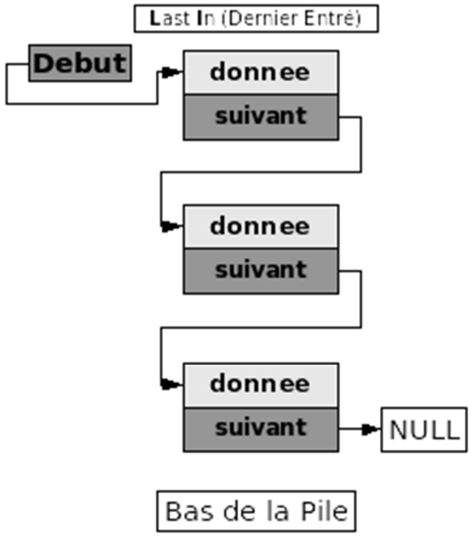


Figure 4.1 – Exemple de pile

Définition 4.4 (arbre binaire)

Un arbre binaire est une structure de données qui peut se représenter sous la forme d'une hiérarchie dont chaque élément est appelé nœud, le nœud initial étant appelé racine. Dans un arbre binaire, chaque élément possède au plus deux éléments fils au niveau inférieur, habituellement appelés gauche et droit. Du point de vue de ces éléments fils, l'élément dont ils sont issus au niveau supérieur est appelé père. La figure 4.2 suivante illustre un arbre binaire.

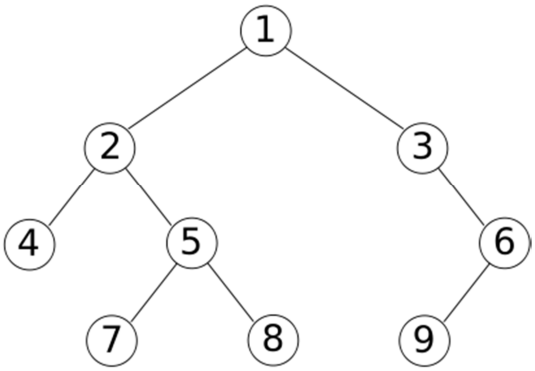


Figure 4.2 – Exemple d'arbre binaire

4.2. Implémentation de la méthode

Dans cette partie nous allons expliquer les détails techniques et de programmation que nous avons utilisés lors de la réalisation de notre application. Nous avons réalisé ce travail en deux parties, la première est une partie en Java qui servira à matérialiser le parseur de la première phase de l'algorithme d'extraction de résumés d'arbres (SAX) ; et cette partie fait appel à la seconde partie qui est en Pascal qui représente le second parseur.

4.2.1. Partie Java

Dans cette partie, on ne va se concentrer que sur les segments de programmation essentiels, à savoir la création du parseur XML (SAX), la sélection d'un seul fichier et son analyse, la sélection d'une collection de fichiers et leur analyse.

4.2.1.1. Création du parseur XML

Le parseur XML est la partie centrale du traitement java, car c'est lui qui effectue l'essentiel du travail en analysant les documents XML. C'est ce parseur qui représente le premier parseur dans l'algorithme d'extraction d'arbres résumés proposé par les auteurs [2]. C'est une instance d'une classe (SimpleContentHandler dans notre cas) qui implémente l'interface ContentHandler qui est l'interface la plus importante qu'implémentent la plupart des applications SAX ; qui sera déclarée comme suit :

```
public class SimpleContentHandler implements ContentHandler
```

Comme toute interface, l'interface ContentHandler contient un ensemble de méthodes qu'il faut implémenter. Dans le cadre de notre travail de programmation, la finalité est d'extraire une forme parenthèsée à partir de la structure arborescente des documents XML à analyser, et de sauvegarder celle-ci dans un fichier (en utilisant un FileWriter, nommé Bassel dans le cas de notre travail) pour pouvoir la réutiliser dans la suite des traitements. Pour ce faire, nous avons implémenté les méthodes de l'interface de telle sorte à obtenir cette forme parenthèsée.

Nous allons détailler certaines des méthodes de l'interface ContentHandler qui nous ont été utiles pour réaliser le parseur :

La méthode setDocumentLocator

Définition du locator qui permet à tout moment pendant l'analyse, de localiser le traitement dans le flux. Le locator par défaut indique, par exemple, le numéro de ligne et le numéro de caractère sur la ligne.

La méthode startDocument

C'est l'événement déclenché au démarrage du parsing d'un document XML. Notre implémentation fait que cette méthode crée une instance du tampon d'écriture PrintWriter qui servira à écrire à l'intérieur du fichier qui sera ouvert en écriture, et renvoie un message indiquant le début de l'analyse du document XML.

```
bassel = new PrintWriter(new FileWriter("nana.txt"),true);

System.out.println("Debut de l'analyse du document");
```

La méthode endDocument

C’est l’événement déclenché à la fin de l’analyse d’un document. Cette méthode écrira dans le fichier un symbole « * » qui indiquera la fin de ce dernier, et renverra un message indiquant la fin de l’analyse, et fermera le fichier en écriture.

```
bassel.print("*");
bassel.close();
System.out.println("Fin de l'analyse du document" );
```

La méthode startElement

C’est l’événement déclenché à la rencontre d’une balise ouvrante. Cette méthode écrira dans le fichier une parenthèse ouvrante ‘(‘ suivie du localName de la balise.

```
bassel.print("( "+localName);
System.out.print("("+localName);
```

Cette méthode parcourra aussi la liste des attributs, et pour chaque attribut rencontré, elle effectuera le même traitement.

La méthode endElement

C’est l’événement déclenché à la rencontre d’une balise fermante par l’analyseur. Cette méthode écrire dans le fichier une parenthèse fermante ‘)’ et renverra un message qui indiquera une parenthèse fermante.

```
bassel.print(")");
System.out.print(")");
```

Pour le reste des méthodes, à savoir :

- startPrefixMapping ;
- endPrefixMapping ;
- characters ;
- ignorableWhitespace ;
- processingInstruction ;
- skippedEntity ;

Elles ont été implémentées par des instructions NOP (No OPeration). On peut implémenter une NOP de différentes manières ; l'une des plus simples :

```
System.out.println("") ;
```

4.2.1.2. Sélection d'un seul fichier XML et son analyse

La sélection d'un fichier se fera en parcourant tout d'abord n'importe quel support de stockage qui serait relié à la machine ; ceci est réalisé par le segment de code suivant :

```
if ( obj == buttonSourceBrowse )
                {
                        int i = fileChooser.showOpenDialog(c);

                        if ( i == JFileChooser.APPROVE_OPTION )
                        {
                                sourceFile =
fileChooser.getSelectedFile().getAbsolutePath();
                                textSourceFile.setText(sourceFile);

                                if ( sourceFile != null  )
                                {
                                        buttonMove.setEnabled(true);
                                }
                        }
                }
```

Une fois le fichier sélectionné, son chemin absolu sera récupéré avec le segment de code [*fileChooser.getSelectedFile().getAbsolutePath()*] et sera utilisé l'analyse du fichier que l'on détaillera plus loin. La sélection d'un fichier aura aussi pour effet d'activer le bouton qui permettra de lancer l'analyse (« parser ») avec le segment de code :

```
if ( sourceFile != null  )
    {
    buttonMove.setEnabled(true);
    }
```

Une fois que le fichier désiré est sélectionné, il faudra définir un fichier de destination où les résultats de l'analyse seront écrits, et cela en saisissant le nom de ce dernier dans un champ de saisie (textdestFile) et en y ajoutant une extension (*.txt). Une fois l'analyse lancée (via le bouton approprié), le nom du fichier de destination sera complété pour créer son chemin absolu ("c:\\Arbre generique\\"+textdestFile.getText()), et une instance du parseur (analyseur) sera créée, pour lancer le parsing du document XML (parser1.parse(sourceFile)).

```
XMLReader parser1 =
XMLReaderFactory.createXMLReader("org.apache.crimson.parser.XMLReaderImpl");

parser1.setContentHandler(new SimpleContentHandler());

parser1.parse(sourceFile);
```

Une fois le parsing effectué correctement, il sera procédé à l'appel de la méthode externe (en Pascal) que l'on détaillera plus loin dans ce chapitre. Cet appel se fait de la manière suivante :

```
Process proc = Runtime.getRuntime().exec( "C:\\execute2.exe" );
```

4.2.1.3. Sélection d'une collection de documents et leur analyse

Le principe est le même que pour un seul document, à la différence que l'on sélectionnera un dossier contenant la collection de documents XML que l'on voudra analyser. Dans ce cas, on ne parcourra que les dossiers, et on ne s'intéressera, plus aux fichiers, et ceci est réalisé grâce au segment de code suivant :

```
JFileChooserfileChooser = new JFileChooser();
fileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
```

La liste des fichiers contenus dans le dossier sélectionné sera enregistrée à son tour dans un fichier avec le segment de code suivant :

```
File[] subfiles = directory.listFiles();
```

A partir de cette liste de fichiers, une boucle sera utilisée pour les parcourir un à un et de les analyser en faisant appel à la méthode externe comme cité précédemment.

Dans ce cas, on calcule le temps que prendra parseur pour analyser toute la collection de fichiers considérée, et pour ce faire on insérera le segment de code suivant avant et après les traitements :

```
Date currentDate = new Date();
long msec = currentDate.getTime();
```

A la fin, le programme procédera au calcul de la différence entre le temps avant et après les traitements, et le résultat sera renvoyé dans un message.

4.2.2. Partie Pascal

C'est le programme auquel fait appel la partie Java que nous avons détaillée précédemment. C'est le code qui représente le second parseur de l'algorithme des auteurs [2].

Son rôle est d'exploiter la forme parenthèsée qui a été extraite des documents XML par le parseur SAX, et d'arriver à créer un fichier récapitulatif des nœuds et de leurs descendances après élimination des redondances possibles.

Comme nous l'avons cité précédemment, c'est la méthode des auteurs [2] que nous avons utilisée pour procéder à l'élimination des répétitions des nœuds, et donc à la création de l'arbre résumé.

Nous allons détailler, une à une, les différentes procédure qui composent ce programme (execute2.pas).

❖ la procédure arbre_xml

Cette procédure peut être considérée comme la procédure centrale du programme, non pas parce qu'elle réalise l'essentiel du travail, mais parce qu'elle contient toutes les autres procédures qui serviront à effectuer les traitements. Avant d'énumérer et de détailler les procédures contenues dans arbre_xml, il est de rigueur de commencer par l'éclaircissement des structures de données manipulées pendant les traitements, en donnant l'essentiel des types de données :

Le type 'enr' : est un enregistrement contenant deux champs :

- le champ 'co' de type char qui contiendra les symboles lus à partir de la forme parnthèsée : (,) , *;
- le champs 'nom' de type string qui contiendra le nom d'une balise lue.

Le type 'pile1' : c'est un enregistrement contenant deux champs :

- le champ 'info1' de type char qui contiendra les symboles de la forme parenthèsée ;
- le champ 'follow' de type lien1 qui est un pointeur servira à pointer vers l'élément suivant.

Ce type permettra de construire la pile syntaxique.

Le type 'lien1' : c'est un pointeur vers un élément de type pile1.

Le type ‘pile2’ : c’est un enregistrement avec 3 champs :

- le champ ‘info2’ de type string qui contiendra le nom d’une balise;
- le champ ‘p1’ de type lien2 qui pointera, au final, vers l’élement fils;
- le champ ‘p2’ de type lien2 qui pointera, au final, vers l’élément frère;

Ce type permettra de construire l’arbre sémantique.

Le type ‘lien2’ : c’est un pointeur vers un élément de type pile2.

Le type ‘table’ : c’est un enregistrement contenant 5 éléments :

- le champ ‘num’ de type integer ;
- le champ ‘name’ de type string ;
- le champs ‘fils’ de type integer ;
- le champs ‘frer’ de type integer ;
- le champs ‘père’ de type integer ;

Ce type sera utilisé lors de la création du fichier récapitulatif final.

La pile syntaxique est une pile qui aidera à se situer dans l’ordre des syboles lus. Quant à l’arbre syntaxique, c’est l’arbre XML proprement dit.

On utilisera également deux pointeurs S et S2 de type lien2 qui serviront à pointer vers les nœuds en cours de traitement. S2 pointera vers le nœud considéré lors du traitement, et S pointera vers la destination du nœud S2 après qu’il soit ordonné.

Voici maintenanat les différentes procédures qui manipulent ces structures de données :

✓ <u>La procédure Reduce1</u>

C’est une procédure qui est appelée pendant les traitement lorsque le champ ‘co’ de la variable ‘inpt’ de type enr contient une parenthèse ouvrante ‘(‘.

Elle effectue les tratements suivants :

- Création d’un nouveau nœud de la pile syntaxique qui contiendra la parenthèse ouvrante et qui sera pointé en tant que tête de pile ;
- Création d’un nœud de l’arbre sémantique, qui contiendra le nom de la balise correspondante.
- Lecture du symbole suivant dans la forme parenthèsée.

✓ La procédure Reduce2

Cette procédure a pour but de rajouter un nœud dans l'arbre sémantique, elle effectue les traitements suivants :

- Duplication du nœud pointé par S2 ;
- Récupération de la descendance de S2 (désordonnée) en tant que fils du nouveau nœud créé ;
- Elimination du nœud pointé par S2 en déplaçant ce dernier ;
- Lien du nœud créé en tant que frère du nœud pointé par S.
- Appel à la procédure Reduce1 dans le cas où le champs 'inpt.co' contient une parenthèse ouvrante.

✓ La procédure Reduce3

Cette procédure sert à détecter une éventuelle duplication de nœuds frères, c'est-à-dire des nœuds qui seraient au même niveaux et qui auraient la même valeur d'étiquette. Elle sera appelée au moment où le nœud de destination (nœud pointé par S) a lui-même déjà un frère. Elle procédera en initialisant un booléen à 'faux' et en le mettant à 'vrai' dans le cas où l'un des nœuds 'frères' du nœud de destination aurait la même étiquette que le nœud considéré. Le pointeur S sera déplacé d'un nœud vers son frère jusqu'à ce qu'il n'y en ait plus ou que l'on rencontre un nœud qui porte la même étiquette.

✓ La procédure Reduce4

Cette procédure, comme Reduce2, rajoute un nouveau nœud dans l'arbre sémantique, et effectuera quasiment les mêmes traitements. La différence est que dans ce cas précis le nœud que l'on cosidère ne possède pas de descendance. Donc la procédure dupliquera le nœud et le reliera à sa destination en éliminant la première occurrence de celui-ci.

La procédure Root

C'est la procédure qui sera appelée pour créer le nœud 'racine' de l'arbre sémantique ; elle interviendra au moment où la valeur de 'inpt.co' sera le symbole '*' , qui signifira la fin du fichier contenant la forme parenthèsée. Elle effectue les traitements suivants :

- Création d'un nouveau nœud (qui deviendra la racine) ;
- Récupération dans le nouveau nœud la valeur du nœud considéré à ce moment ;
- Récupération de la descendance du nœud considéré ;

✓ La procédure Reduce5

L'appel à cette procédure survient lorsque la procédure Reduce3 retourne qu'il y a duplication d'un nœud au même niveau hiérarchique. Son rôle est d'effectuer l'élimination de toute répétition de nœud, tout en évitant une quelconque perte d'information, c'est-à-dire éliminer un nœud redondant tout en récupérant son éventuelle descendance et ses frères.

Cette procédure est récursive, et fonctionne comme tel :

- Lorsque le nœud redondant à éliminer l'a ni frère ni fils, il sera procédé à son élimination direcement.
- Lorsque le nœud redondant à éliminer possède une descendance, si le premier frère, c'est-à-dire le nœud qui a la même étiquette mais que l'on souhaite garder, n'a pas de descendance, la descendance du nœud redondant sera récupérée vers le premier frère et il sera supprimé. Dans le cas où le premier frère possède également un fils, un appel récursif à cette procédure vendra effectuer les mêmes traitements sur les deux fils.
- Lorsque le nœud redondant à éliminer possède un frère, il faudra récupérer ce dernier si le nœud premier frère n'a aucun frère ou qu'il ne possède pas la même étiquette que l'un des frères de celui-ci. Dans le cas contraire, il sera ignoré et éliminé avec le nœud redondant en récupérant sa descendance de la manière précédemment citée.

✓ La procédure Imprimer

Cette procédure est celle qui permet d'écrire dans le fichier final qui contiendra la récapitulatif des nœuds et de leur descendance. Elle exploitera l'arbre « résumé » obtenu par tous les traitements précédents. Elle utilise une procédure nommée Taille pour calculer le nombre de descendants d'un nœud, et utilise aussi des structures de données de type 'table' pour exploiter les informations sur les nœuds. Le fichier en sortie de cette procédure contiendra pour chaque nœud de l'arbre résumé :

- Son numéro ;
- Le numéro de son fils, et si le nœud n'a pas de fils la valeur sera (-1);
- Le numéro de son frère, et s'il n'a pas de frère la valeur sera (-1) ;
- Le numéro de son père, et dans le cas du nœud racine la valeur (0) est attribuée en tant que numéro du père.

✓ La procédure ImprimerArbre

Cette procédure sert à faire appel à la procédure imprimer, avec un pointeur qui référence le nœud racine de l'arbre résumé obtenu.

✓ Corps de la procédure centrale Arbre_xml

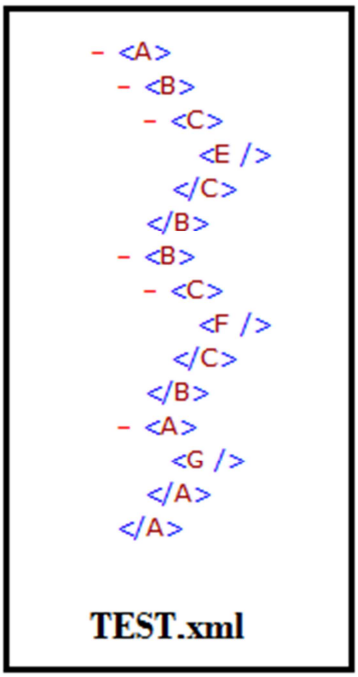
C'est la partie qui fera appel à toutes les autres procédures énumérées précédemment. Elle effectue les traitements suivant :

- Assigner le nom 'code ' au fichier contenant la forme parenthèsée ;
- Charger la table de précédence d'opérateurs en mémoire ;
- Création du premier nœud de la pile syntaxique avec la valeur '*'en étiquette;
- Tant que la valeur du sommet de la pile syntaxique est différente de '*'ou bien la valeur de inpt.co est différente de '*', il comparera selon la table de précédence d'opérateurs les deux valeurs dans l'ordre :
- Si le résultat de la comparaison est '<' ou '=' il sera procédé à un décallage en créant un nœud de la pile syntaxique contenant la valeur de inpt.co. Dans le cas unique de quand la précédence est '<' un nœud de l'arbre sémantique est créé avec la valeur de inpt.nom comme étiquette. Une nouvelle lecture de code sera faite.
- Si le résultat de la comparaison est '>', il y aura dépilement deux fois dans la pile syntaxique et selon les cas, il y aura appel aux procédures qui effectueront le reste des traitements afin d'obtenir le résultat escompté c'est-à-dire l'arbre résumé et le fichier récapitulatif.

4.2.3. Expérimentation

Nous allons dérouler une exécution des traitements sur un exemple de fichier XML de notre création, en donnant l'essentiel des étapes en passant par toutes les parties.

Nous utilisons le fichier TEST.xml suivant qui contient tous les cas possibles traitables dans notre travail, à savoir la duplication et l'imbrication de nœuds.



Le traitement de ce fichier passera par les étapes principales dont nous avons parlé précédemment. Tout d'abord il sera procédé à l'extraction de la forme parenthèsée à partir de la structure arborescente du fichier XML, puis à la construction de l'arbre (résumé) à partir de la forme parenthésée, et enfin la création du fichier récapitulatif de ce document XML à partir de l'arbre.

Après le chargement du fichier en mémoire, le parseur SAX prendra le relais et le lira de son début à sa fin et en déclenchant les routines appropriées à chaque rencontre d'une balise ouvrante, ou fermante, etc., tout en construisant la formé parenthèsée qui nous intéresse.

Voici la formé parenthèsée extraite à partir de TEST.xml :

(A (B (C (E))) (B (C (F))) (A (G)))

Après l'extraction de cette forme parenthésée qui sera enregistrée dans un fichier, il sera fait appel, comme cité précédemment, à une méthode externe réalisée en Pascal afin de lire cette forme parenthésée symbole par symbole et de construire l'arbre résumé du fichier XML au préalable, puis de créer le fichier récapitulatif.

Nous avons réalisé cet arbre sous forme d'arbre binaire dont les nœuds seront créés un à un suivant les algorithmes programmés, et cela en procédant à l'élimination des nœuds

redondants (du même niveau) et en garantissant qu'aucune information n'est perdue. Cette étape donnera naissance en mémoire à l'arbre résumé dans la Figure 4.3 suivante.

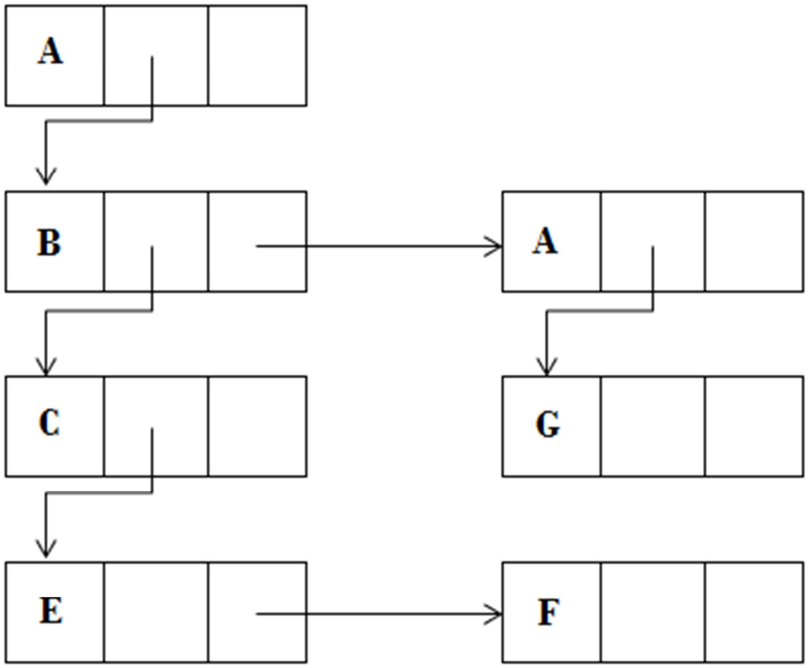


Figure 4.3 –Arbre binaire résumé du document TEST.xml en mémoire

Cette forme arborescente montre clairement chacun des éléments du document XML. On remarque bien l'élimination du nœud redondant 'B' et la récupération de son fils qui est le nœud 'F' en tant que frère du nœud 'E' qui est le fils du premier nœud 'B'. On remarque aussi que le nœud 'A' apparait deux fois dans l'arborescence car il apparait à des niveaux différents (imbrication de nœuds).

Une fois cet arbre construit en mémoire, il sera procédé à la dernière partie du traitement qui démarrera à partir de la racine et lira tous les nouds un à un pour construire un fichier récapitulatif de ces derniers. Ce fichier portera le même nom que le fichier XML source et dans notre cas (pour TEST.xml) il sera construit comme dans la figure 4.4 qui suit.

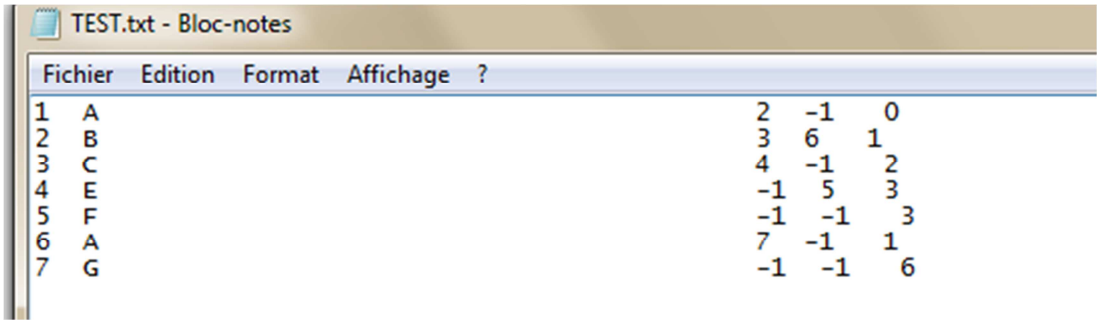


Figure 4.4 – le fichier récapitulatif TEST.txt

Ce fichier donne pour chaque nœud, dans l'ordre, son numéro, son nom, le numéro de son fils, le numéro de son frère et le numéro de son ancêtre.

4.2.4. Exécution illustrée

Nous allons donner un exemple d'exécution de notre application en passant par tous les cas possibles.

Nous commencerons par le parsing d'un seul document XML, et nous finirons avec le parsing d'une collection de documents.

Tout d'abord, il faudra veiller à ce que toutes les DTDs qui concernent les documents susceptibles d'être analysés soient présentes, comme indiqué dans lesdits documents, dans le disque C:/ contenant le système.

Au lancement de l'application, il sera affiché à l'écran une fenêtre proposant un choix entre le parsing d'un seul document XML et d'une collection de documents XML. Cette fenêtre est illustrée dans la Figure 4.5 suivante.

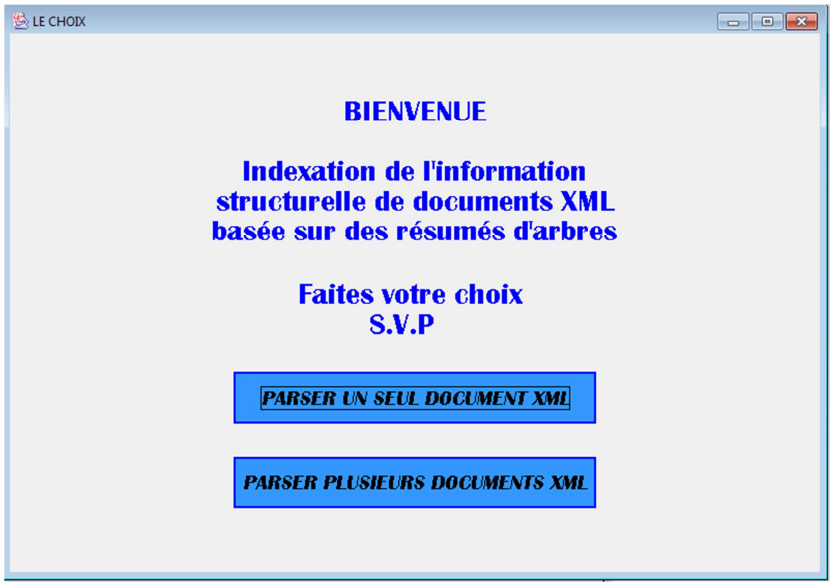


Figure 4.5 – Fenêtre principale de l'application

4.2.4.1. Parsing d'un seul document

Pour choisir de parser un seul document XML, il faut cliquer sur le bouton correspondant, dont le label est la mnémonique de l'action. Une fois ce choix fait, la fenêtre présentée dans le Figure 4.6 s'affichera à l'écran pour demander de confirmer ce choix.

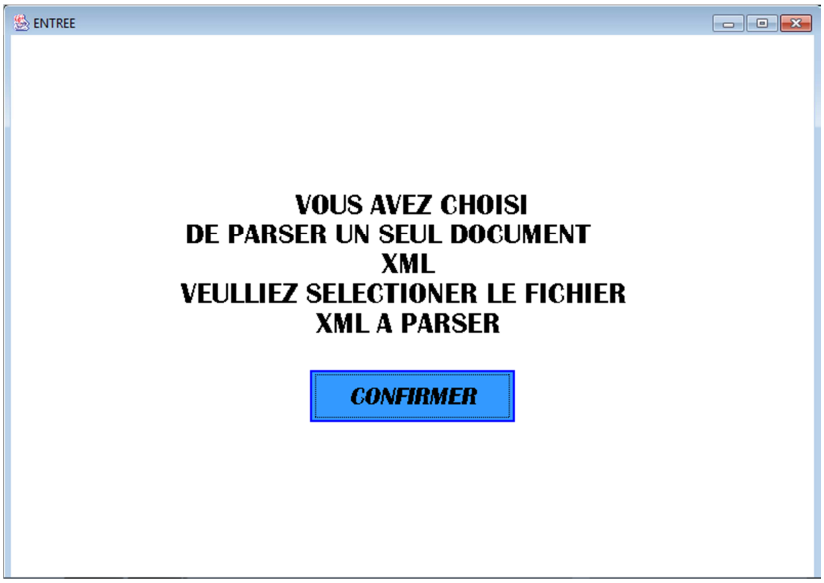


Figure 4.6 – Confirmation du choix

Une fois le choix confirmé en cliquant sur le bouton de confirmation, une nouvelle fenêtre s'affichera à l'écran qui se compose principalement d'un bouton pour parcourir les périphériques de stockage de la machine pour sélectionner le fichier XML que l'on désire

analyser, d'un bouton qui permettra d'ouvrir un éditeur de texte tel que 'bloc-notes' pour saisir un nouveau document XML et d'un bouton pour lancer les traitements d'analyse proprement dits (ce bouton n'est pas encore actif à cette étape). La Figure 4.7 suivante illustre cette fenêtre.

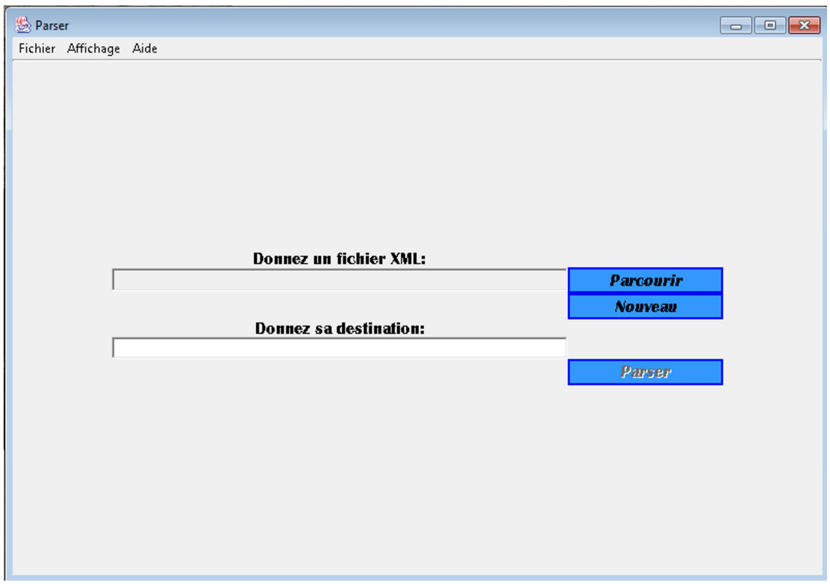


Figure 4.7 – Fenêtre de parsing d'un seul document

Afin de sélectionner un document que l'on désire parser, on cliquera sur le bouton 'parcourir' qui mènera vers un browser qui permettra l'accès et le parcours des disques comme illustré dans la Figure 4.8 suivante

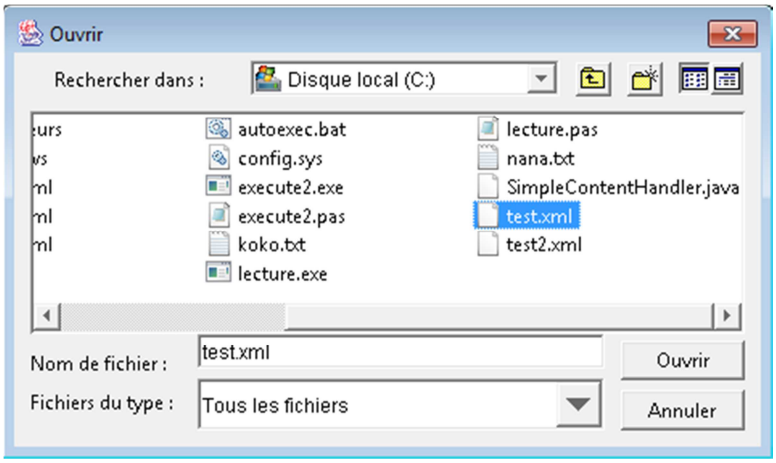


Figure 4.8 – Sélection d'un fichier XML

Une fois que le fichier que l'on veut parser est sélectionné, il faudra introduire le nom du fichier de destination (dans le champ réservé à cela) qui sera le fichier récapitulatif de ce

document. On remarquera l'activation du bouton 'Parser' qui lance les traitements. Comme illustré dans la Figure 4.9 suivante.

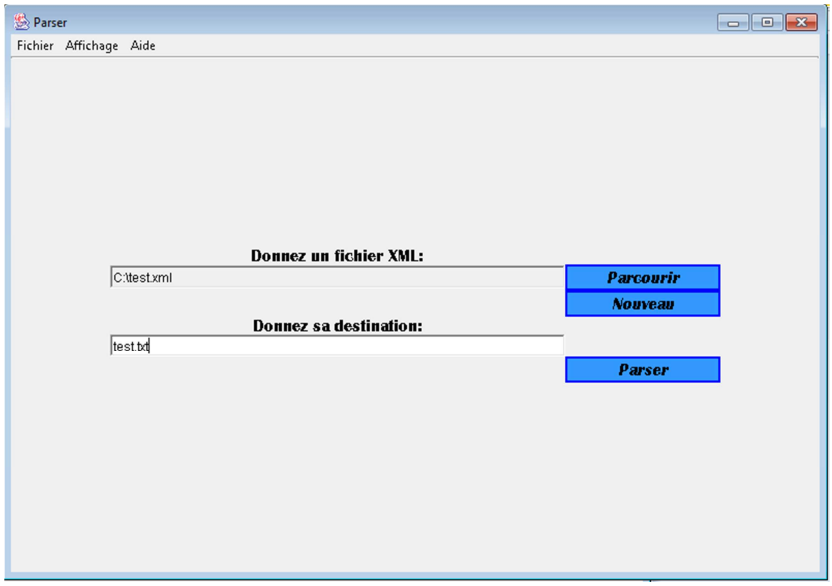


Figure 4.9 – Lancement du parsing

Une fois le parsing lancé, il sera procédé à l'analyse du document XML sélectionné suivant les traitements appropriés. La Figure 4.10 suivante montre ce qui sera affiché en sortie par le langage de programmation (Java), comme les messages et la forme parenthèsée de l'arborescence du document XML.

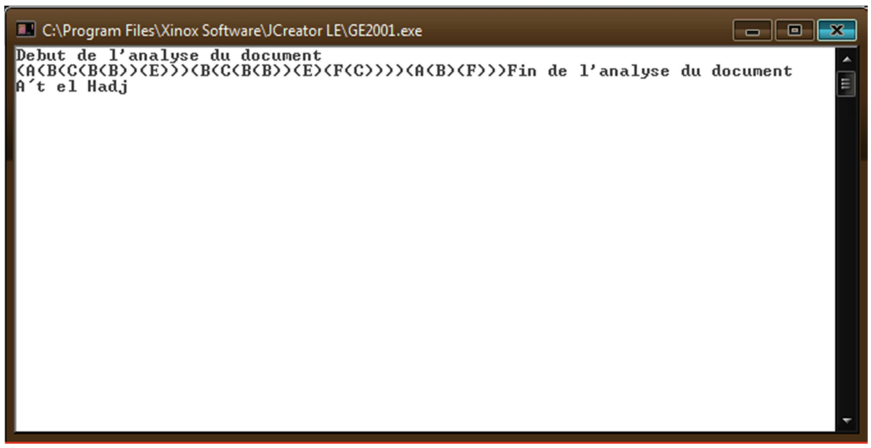


Figure 4.10 – Messages de sortie.

Après le parsing du document XML et la création du fichier récapitulatif de sortie, un message prévenant du bon déroulement de l'analyse s'affichera à l'écran, comme illustré dans la Figure 4.11 suivante.

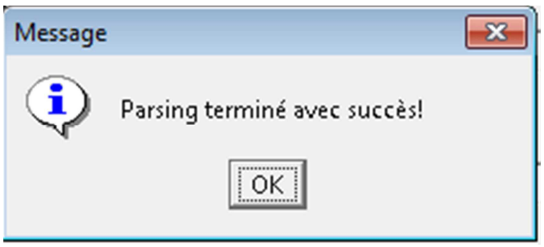


Figure 4.11 – Message de fin des traitements

4.2.4.2. Parsing d'une collection de documents

Pour parser une collection de documents, il faut cliquer sur le bouton correspondant dans la fenêtre principale. Une fois ce choix effectué, une fenêtre s'affichera à l'écran afin de confirmer ce choix, comme illustré dans la Figure 4.12 suivante.

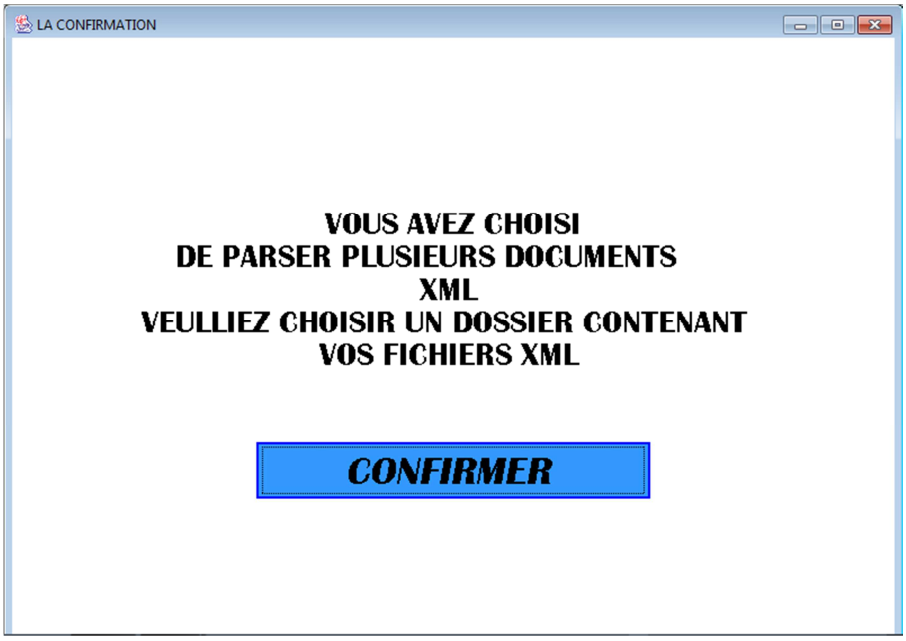


Figure 4.12 – Fenêtre de confirmation

Quand le choix est confirmé en cliquant sur le bouton de confirmation, la fenêtre illustrée dans la figure 4.13 s'affichera à l'écran.

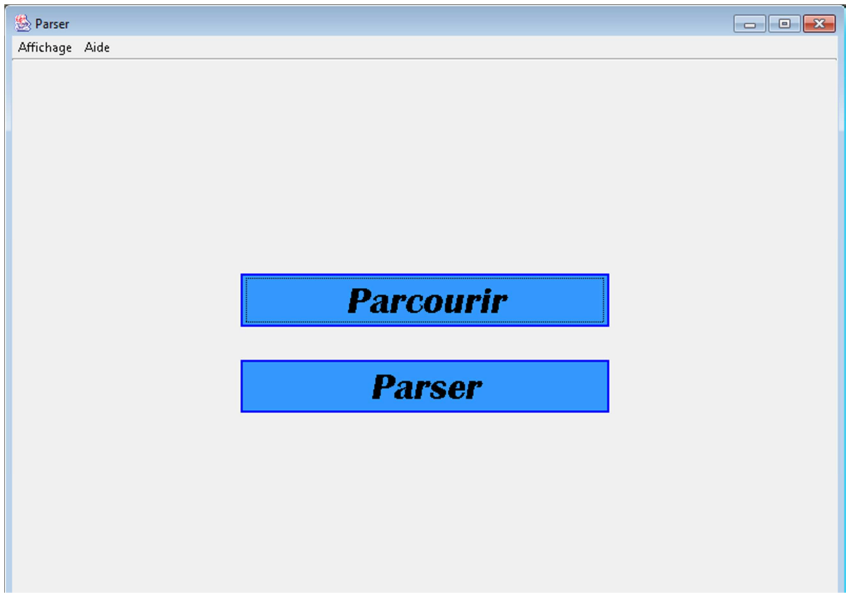


Figure 4.13 – Fenêtre de parsing de plusieurs documents XML

Cette fenêtre est composée principalement de deux boutons. Le premier bouton permet comme pour un seul document de parcourir les périphériques de stockage, à la différence que dans ce cas le browser ne s'intéressera qu'aux dossiers, ceci permettra de sélectionner un dossier contenant la collection de documents XML que l'on voudra analyser. Dans notre cas nous sélectionnerons un dossier (TEST XML) qui contient 50 documents XML. Ceci est montré dans la Figure 4.14 qui suit.

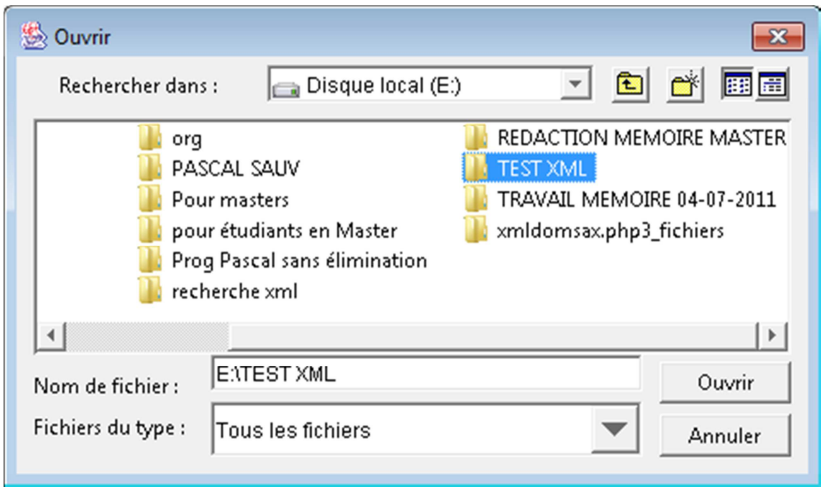


Figure 4.14 – Browser pour sélectionner un dossier

Une fois le dossier sélectionné, on lancera l'analyse par le biais du bouton 'Parser'. Comme pour l'analyse d'un seul document, l'analyse de chaque document de la collection donnera lieu à des messages de sorties ainsi que de la forme parenthèsée de chacun, comme illustré dans la Figure 4.10 précédente.

A la fin du parsing des documents contenus dans le dossier sélectionné, il sera affiché à l'écran un message indiquant le bon déroulement de l'analyse de tous les documents, ainsi que le temps pris par le traitement de la collection, comme illustré dans le Figure 4.15. Ce calcul de temps est réalisé dans un but comparatif, que nous verrons plus loin dans ce chapitre.

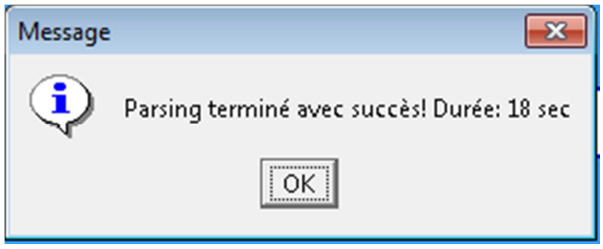


Figure 4.15 – Message de fin de traitements avec affichage de la durée

Les fenêtres d'analyse d'un document et d'une collection de documents ont en commun une barre d'outils, comme illustré dans la Figure 4.16 suivante.

Figure 4.16 – Barre d'outils

Cette barre d'outils est composée de trois boutons : Fichier, Affichage et Aide.

Le bouton Fichier donne un menu déroulant qui contient plusieurs actions (comme montré dans la Figure 4.16 précédente), parmi celles-ci on a la création de nouveau document XML, la possibilité d'enregistrer le fichier XML créé sur disque, et de quitter.

Le bouton afficher permet d'afficher l'arborescence d'un document XML que l'on désire visionner. Quant au bouton aide, il donne accès à des informations concernant l'application, comme dans la Figure 4.17 suivante.

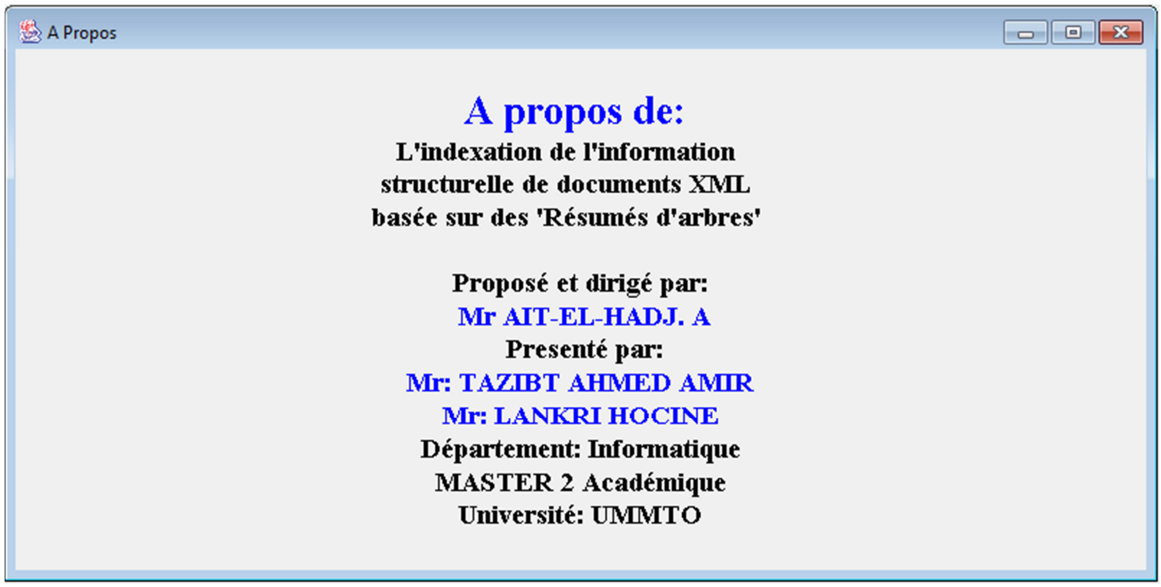


Figure 4.17 – A propos de l'application

Pour quitter l'application, on a la possibilité d'utiliser la fonction 'Quitter' du menu déroulant du bouton Fichier, ou de simplement cliquer sur la croix rouge des fenêtres. En effectuant l'une de ces actions, une fenêtre s'affichera pour demander confirmation de la fermeture de l'application, comme dans la Figure 4.18 qui suit.

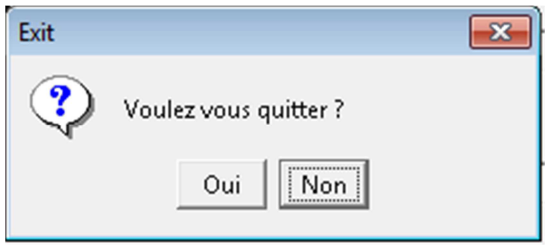


Figure 4.18 – Confirmation de fermeture de l'application

4.2.5. Intérêt de la méthode

Dans le cadre de notre travail de programmation, le défi qui nous a été demandé de surmonter est l'extraction de « résumés d'arbres » à partir des structures arborescentes des documents XML que l'on manipule, pour au final obtenir les fichiers récapitulatifs de ces documents qui seront utiles dans d'autres travaux. Pour extraire des « arbres résumés », il est nécessaire de faire une lecture du contenu des documents XML afin d'en tirer l'essentiel des nœuds en éliminant les redondances des nœuds frères (au même niveau) et en évitant toute perte d'information susceptible d'affecter la cohérence du document (en récupérant la descendance de chaque nœud éliminé si possible).

Il peut résulter plusieurs avantages de l'utilisation d'arbres résumés, selon les cas. Nous allons effectuer une comparaison, selon les critères les plus évidents et les plus intéressants, entre l'utilisation des résumés d'arbres, et celle des arborescences complètes des documents XML.

Voici pour un document XML, son résumé d'arbre et sa structure arborescente complète dans la Figure 4.19 suivante. Nous avons utilisé la même représentation d'arborescence que lors de la programmation. Autrement dit, nous utilisons un arbre binaire, dont chaque nœud a pour fils gauche son descendant, et pour fils droit son frère.

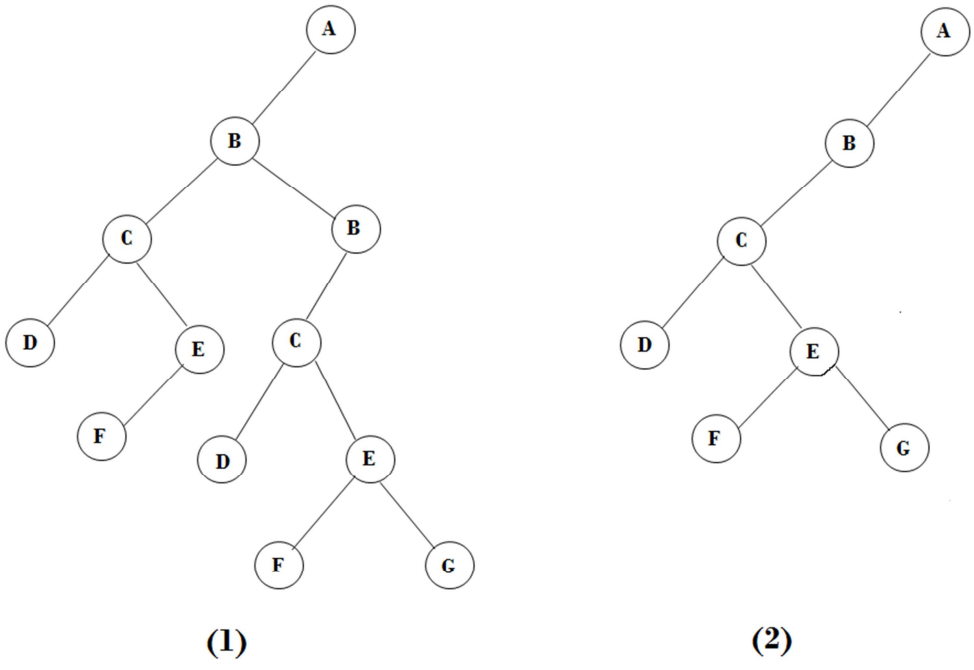


Figure 4.19 – Arborescente entière (1) et résumé d'arbre (2)

Cette figure met en évidence les différences entre les deux structures arborescentes. On remarque déjà pour un très petit document, une diminution de près de la moitié du nombre des nœuds de l'arbre, ce qui pourrait engendrer un gain conséquent d'espace mémoire, surtout en ce qui concerne une collection documentaire contenant quelque milliers de documents qui seraient à leur tour volumineux.

Nous avons réalisé une expérimentation sur des parties du corpus documentaire XML ACM SIGMOD, contenant chacune une cinquantaine de documents, dans un but comparatif. Dans un premier cas, nous avons occulté les traitements qui éliminent les redondances, et dans l'autre nous avons laissé nos traitements tel quel. Tout d'abord, nous avons réalisé des tests sur un corpus de documents XML contenant des documents XML moyennement volumineux ou de petit volume, et dans ce cas, nous avons pu remarquer que sur ce corpus l'élimination des redondances éventuelles prenait légèrement plus de temps que de ne pas les éliminer, une différence allant de 1 à 1,5 secondes maximum (soit un gain de moins de 10% de temps) pour analyser et extraire les formes arborescentes et les fichiers récapitulatifs de la cinquantaine de documents XML. Cependant, la différence la plus marquante, est dans le cas de l'utilisation d'un corpus documentaire contenant de très gros documents XML. On a noté dans ce cas des différences pouvant avoisiner les 8 ou 10 secondes, soit entre 40% et 50% de gains de temps pour le même nombre de documents quand dans le cas précédent, ce qui peut s'avérer énorme dans le cas d'autres corpus de documents contenant des centaines ou même des milliers de documents XML tous volumineux. Ce que l'on a pu noter également, c'est que dans certains cas, comme pour un document/corpus Air France ou Renault qui sont de très gros documents XML, le fait d'ignorer l'élimination des redondances d'information provoque souvent des dépassements de capacité mémoire (Memory Overflow) au moment d'effectuer les traitements de la création de l'arbre en mémoire, ou même de l'extraction du fichier récapitulatif, alors que pour les mêmes documents, l'élimination des redondances a donné la possibilité d'avoir les résultats escomptés le plus normalement.

Cette expérimentation nous aura prouvé que même si dans des cas minimes, l'élimination de la redondance consomme plus de temps que de ne pas éliminer les répétitions, il est toujours préférable d'éliminer les redondances éventuelles, car ceci offrira une plus grande garantie d'avoir de meilleurs résultats et plus cohérents, et surtout dans les cas les plus intéressants d'avoir une plus grande rapidité de traitements, ce qui est un autre gros avantage.

Les structures d'arbres résumés offrent aussi un aspect très important qui est la fiabilité, car cette forme de représentation garantit qu'aucune information ne sera perdue et offrira une représentation concise du document XML.

Enfin, l'utilisation de résumés d'arbres permettront de donner naissance à des fichiers récapitulatifs plus courts et concis, et donc mieux compréhensible par l'être humain. La Figure 4.20 suivante montre le contenu du fichier récapitulatif en utilisant des structures arborescentes complètes, et la Figure 4.21 montre le contenu de ce fichier avec un arbre résumé.

Figure 4.20 – Fichier récapitulatif en utilisant une structure d'arbre complète

Figure 4.21 – Fichier récapitulatif en utilisant un résumé d'arbre

On remarque bien à partir des deux figures précédentes que l'élimination des redondances donne naissance à des fichiers récapitulatifs plus précis. Ces fichiers récapitulatifs seront utilisés dans le cadre de la classification structurelle de documents XML, qui n'entre pas dans le cadre de notre travail, mais c'est dans ce domaine que l'impact de la précision et de la diminution des fichiers récapitulatifs se fera réellement ressentir au niveau de la vitesse des traitements.

4.2.6. Environnement de programmation

Nous allons détailler brièvement l'environnement et les différents outils et langages manipulés lors de la programmation de notre application.

4.2.6.1. Le Java

Java est le nom de marque d'une technique informatique développée par Sun Microsystems : la « technologie Java ».

Java est utilisé dans une grande variété de plates-formes depuis les systèmes embarqués et les téléphones mobiles, les ordinateurs individuels, les serveurs, les applications d'entreprise, les superordinateurs, etc.

Défini à l'origine comme un langage, « Java » a évolué au court du temps pour devenir un ensemble cohérent d'éléments techniques et non techniques. Ainsi, la technologie Java regroupe :

Des standards (la plate-forme Java) définis par le Java Community Process (JCP), en trois éditions :

- Java SE (standard edition) ;
- Java EE (enterprise edition), s'appuyant sur Java SE ;
- Java ME (micro edition), indépendante des deux précédentes.

Des logiciels (langages informatiques, bibliothèques, frameworks, serveurs d'application, outils d'aide au développement), dont :

- Des implémentations (concurrentes) de ces standards
- Un écosystème d'autres logiciels s'appuyant sur tout ou partie de ces standards, voire leur faisant concurrence

Java est un des termes les plus connus du monde de l'informatique et de l'Internet, que ce soit des professionnels comme du grand public.

4.2.6.2. La classe SWING (Java)

Swing est une bibliothèque graphique pour le langage de programmation Java, faisant partie du package Java Foundation Classes (JFC), inclus dans J2SE. Swing constitue l'une des principales évolutions apportées par Java 2 par rapport aux versions antérieures.

Swing offre la possibilité de créer des interfaces graphiques identiques quel que soit le système d'exploitation sous-jacent, au prix de performances moindres qu'en utilisant Abstract Window Toolkit (AWT). Il utilise le principe Modèle-Vue-Contrôleur (MVC, les composants Swing jouent en fait le rôle du contrôleur au sens du MVC) et dispose de plusieurs choix d'apparence (de vue) pour chacun des composants standards.

4.2.6.3. Pascal

Pascal est un langage de programmation impératif qui se caractérise par une syntaxe claire, rigoureuse et facilitant la structuration des programmes. Cette clarté et cette rigueur font que Pascal était encore récemment souvent utilisé dans l'enseignement.

En dehors de la syntaxe et de sa rigueur, le langage Pascal possède de nombreux points communs avec d’autres langages connus (Le C par exemple). Le langage Pascal de base était conçu à usage purement éducatif et était assez limité (pas de chaînes de caractères, par exemple), mais les développements qu'il a connus en ont fait un langage complet et efficace.

Les versions actuelles de Pascal, utilisées hors du monde éducatif, sont des extensions telles que Turbo Pascal (mode texte), Object Pascal (programmation objet), et Delphi (fenêtré). On peut programmer en Pascal sous la quasi-totalité des systèmes d’exploitation.

Pascal est connu pour avoir permis d'élaborer des logiciels assez renommés comme une partie des premiers systèmes d'exploitation du Macintosh. Sa syntaxe a aussi été reprise par d'autres langages (Ada, Modula-2, …).

4.2.6.4. Jcreator

Pour programmer en Java, nous avons utilisé Jcreator. Jcreator est un IDE très puissant pour Java. C'est l'environnement de programmation Java le plus adéquat pour tout programmeur aimant programmer de la meilleure et la plus efficace des manières. Il est plus rapide, plus efficace et plus fiable que la plupart des autres IDE Java. C'est pour cela que c'est l'outil parfait pour les programmeurs de tout niveau, du novice au plus expérimenté.

Jcreator fournit aux utilisateurs une large palette de fonctionnalités telles que la gestion de projets, des modèles de projets, complément de code, interface de débogage, un éditeur avec coloration syntaxique, des assistants et une interface utilisateur entièrement personnalisable.

Avec Jcreator, on peut directement compiler ou exécuter un programme Java sans activer le document principal au préalable. Il va automatiquement trouver le fichier avec la méthode principale ou le fichier HTML contenant l'applet Java, il démarrera donc l'outil approprié.

Jcreator est entièrement écrit en C++, ce qui le rend rapide et efficace comparé aux éditeurs/IDE basés Java.

4.2.6.5. Microsoft Windows 7

Windows 7 (précédemment connu en tant que Blackcomb et Vienna) est le dernier en date des systèmes d'exploitation de la société Microsoft et successeur de Windows Vista sorti en 2009.

Cette nouvelle version de Windows reprend l'acquis de Windows Vista tout en apportant de nombreuses modifications, notamment par divers changements au niveau de l'interface et de l'ergonomie générale, un effort particulier pour la gestion transparente des machines mobiles et le souci d'améliorer les performances globales du système (fluidité, rapidité d'exécution même sur des systèmes moins performants, tels les netbooks) par rapport à son prédécesseur.

En identifiant cette nouvelle mouture par son numéro de version (il s'agit de la septième version de Windows), Microsoft renoue avec une logique abandonnée depuis Windows 4.2 et Windows NT 4.0. La tradition voulait jusqu'ici que les versions de Windows soient identifiées par référence à l'année de sortie (Windows 95…) ou par une appellation ad hoc (Windows XP ou Windows Vista). Néanmoins, Windows 7 se base sur le noyauNT 6.1.

4.2.6.6. Corpus XML ACM SIGMOD

*Nous avons utilisé dans notre expérimentation un corpus XML de ACM SIGMOD (*Special Interest Group on Management of Data). Ce corpus est composé d'une cinquantaine de documents XML et de 4 DTDs. Ce corpus concerne les articles publiés par ACM SIGMOD.

Conclusion

Dans ce chapitre nous avons exploré les différentes possibilités de notre application, en passant par ses différentes méthodes et leurs détails techniques, des exemples significatifs d'exécution, des illustrations et l'environnement de programmation. Nous avons également montré l'interêt de notre contribution dans le domaine de la recheche d'information sur le Web.

Conclusion générale

Synthèse

Les travaux présentés dans ce mémoire se situent dans le contexte de la recherche d'information, et plus particulièrement dans le cadre de l'indexation des documents semi-structurés.

Un processus d'indexation de la recherche d'information structurée doit concilier le contenu et la structure d'un document pour pouvoir répondre efficacement aux besoins en information des utilisateurs. L'unité retournée par le système de recherche d'information ne doit plus être le document en entier mais des fragments de celui-ci qui se focalisent sur les besoins de l'utilisateur. Pour ce faire, la solution préconisée dans notre travail est l'indexation, et plus particulièrement l'indexation de l'information structurelle dans le cadre de documents XML, qui est le formalisme de base sur lequel nos travaux se sont portés. Cette solution prend en considération cette notion introduite par le formalisme XML, qui est la structure, et non plus seulement le contenu comme dans le cas des documents plats, et doit répondre aux préoccupations suivantes :

- Quelle est la manière la plus judicieuse de représenter l'information structurelle d'un document semi-structuré de type XML ?
- Quelle est l'approche d'indexation de l'information structurelle de documents XML la plus efficace ?

Il existe plusieurs façons de représenter l'information structurelle d'un document semi-structuré, de type XML par exemple. Cependant, la manière la plus simple et compréhensible de représenter cette information structurelle est bien de donner son arborescence. Cette forme arborescente servira à bien cerner les différentes balises, et de bien distinguer la hiérarchie, car chaque nœud et sa descendance seront clairement visibles.

Ce type de représentation peut donner lieu à différentes approches d'indexation de l'information structurelle. En revanche, l'approche que nous avons choisie est l'approche

d'indexation basée sur le formalisme des « arbres résumés ». Ce formalisme est plus générique que la représentation de l'arborescence complète du document XML, car elle permet d'avoir une structure plus concise en éliminant toutes les redondances d'information, tout en évitant une perte d'information. Dans notre travail nous avons choisi l'approche de [2] pour extraire des résumés d'arbres à partir de documents XML.

Nous avons choisi cette approche proposée par [2], car elle nous a semblé la moins contraignante, et la plus sure afin d'arriver à de bons résultats qui ne compromettraient pas la cohérence des documents XML. Cette approche que l'on a choisie répond aux questions posées précédemment.

Cependant, l'étude détaillée du comportement de l'approche que l'on a choisie, une fois implémentée, nous a montré que cette approche est beaucoup plus utile dans le cas de traitement de collections de documents XML qui seraient volumineux, car elle permet de gagner en performances et en temps, et garantit des résultats pertinents et cohérents. Cependant, dans le cas de documents XML de petits volumes, cette approche n'offre pas un grand gain en temps ou en espace mémoire consommé, et peut même dans certains cas s'avérer consommer plus de ressources temporelles par rapport à une autre approche, mais sans être un handicap.

Néanmoins, nous estimons avoir atteint les objectifs que nous nous somme fixés, en présentant une implémentation de l'approche d'indexation par des résumés d'arbres selon [2] qui nous a été proposée dans le cadre de ce mémoire, et qui pourra servir dans le futur dans le but d'améliorer la recherche d'information structurée, et particulièrement dans le cas de documents semi-structurés de type XML, et aussi de faciliter le processus de classification qui peut être un débouché à ce travail.

Perspectives

Les perspectives que l'on peut envisager dans le cadre de notre travail sont plus d'ordre technique. Nous citerons quelques perspectives qui pourront apporter du positif à notre travail.

- ✓ Tout d'abord, l'utilisation du langage Java pour programmer une partie de notre application a ajouté une certaine lourdeur aux traitements. Pour remédier à cette lourdeur, on peut, au lieu d'utiliser Java, utiliser un langage tel que le C (C++), ou bien un autre langage tel que Pascal Objet, pour leur côté programmation objet. Cependant, le langage Java a son utilité ici, car grâce à ses fonctionnalités et à la richesse de ses bibliothèques, c'est le langage le plus adapté au Web de nos jours, et c'est le domaine dans lequel intervient notre application.

- ✓ Enfin, la dernière perspective concerne la dépendance entre le code d'implémentation et les documents XML et leurs DTDs. Il serait intéressant et utile d'apporter une certaine indépendance entre l'emplacement du code d'implémentation de la méthode et l'emplacement des documents XML et leurs DTDs respectives. Ceci apportera plus de flexibilité et de liberté aux utilisateurs de notre application qui ne seraient plus obligés d'avoir les documents XML et leurs DTDs obligatoirement au même endroit que le code.

Bibliographie

[1] Termier, A., (2004). *Extraction d'arbres fréquents dans un corpus hétérogènes de données semistructurées.* Paris, France.: Université Paris Sud 11 Orsay.

[2] Aïtelhadj, A., Mezghiche, M., & Souam, F. (2009). RI structurée, RI et XML, RI précise. *6eme Conférence en Recherche d'Information et Applications Coria'2009*, (pp. 301-317). Presqu'île de Giens, France.

[3] Brillant, A. (2007). *XML cours et exercices.* Paris, France: Editions Eyrolles.

[4] Gutierrez, A., Motz, R., & Viera, D. (2000). Building databases with information extracted from web documents. *international conference of the Chilean computer sciences society*, (pp. 41-49).

[5] J, Yang., & X., Chen. (2002). A Semi-Structured Document Model for Text Mining. *Computer Science and Technology (Journal of)*, 603-610.

[6] Zaki, M. J. (2002). Efficient mining frequent trees in a forest. In Proceedings of the 8th ACM. *International Conference on Knowledge Discovery and Data Mining* (pp. pp 71-80). Edmonton, Alberta, Canada: KDD.

[7] J.P., Yoon., & BitCube, a. (2001). A Three-Dimensional Bitmap Indexing for XML Documents. *JIIS*, 241-254.

[8] P., Kilpeläinen. (1992). *Tree Matching Problem with Applications to Structured Text Databases.* Helsinki: .

[9] Sauvagnat, Karen. (2005). *Modèle flexible pour la Recherche d'Information dans des corpus de documents semi-structurés.* Toulouse, France: Université Paul Sabatier Toulouse.

[10] T, Dalamagas., T, Cheng., K-J, Winkle, & T.K., Sellis. (2006). A methodology for clustering XML. *Information Systems*, 187-228.