Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Saad Dahlab Blida
Centre de Recherche sur l'Information Scientifique et Technique (CERIST)

Mémoire de fin d'études

Pour l'obtention du diplôme de Master en Informatique
Option : Ingenierie des Logiciel

Thème :

Agrégation des résultats dans la recherche d'information XML

Présenté par :
Benaouda-Zouaoui Khalil
Ayadi Ramzy

Dirigé par :
Promoteur : Mlle BOUSTIA .N
Encadreur : M^me^Bessai F.Z

Jury :
Président — Cherif Zahar
Examinateur 1 — Chorfa
Examinateur 2 — Ameur

RÉSUMÉ

La recherche d'information agrégée permet, en réponse à une requête, d'agréger des granules d'information provenant de plusieurs sources et de renvoyer à l'utilisateur un ensemble d'informations bien organisées. Le résultat agrégé est une alternative à la traditionnelle liste de documents répondant chacun à une partie du besoin utilisateur. Nous nous intéressons particulièrement dans cet article à la recherche agrégée, qui se focalise sur les granules d'information. Nous présentons une approche d'agrégation des résultats répondant à une requête d'utilisateur. Afin de pallier ces lacunes une approche basée sur des algorithmes de recherche puissants, exacts et efficaces en utilisant Lucene, Le un moteur de recherche textuelle Open Source fourni par la fondation Apache a été élaborée. On présente une nouvelle stratégie de classification utilisant un principe de compétition et de coopération pouvant remédier aux aléas liés aux classifications classiques.

Mots-clés : Recherche d'Information, documents structurés, XML, Indexation, agrégation, recherche d'information agrégée, agrégation des résultats.

Abstract

The aggregated information can research in response to a request to aggregate information from multiple sources granules and return to the user a set of well-organized information. The aggregate result is an alternative to the traditional list of documents each addressing some of the user requirements. We are particularly interested in this article to research associate, which focuses on information granules. We present an approach to aggregation of results in response to a user request. To overcome these shortcomings based on powerful algorithms, accurate and effective search using Lucene approach the one source search engine Open Text provided by the Apache foundation has been developed. We have a new classification strategy using a principle of competition and cooperation can address the hazards associated with conventional classifications.

Keywords: Information Retrieval, structured documents, XML, Indexing, aggregation, information retrieval aggregate, aggregation of results.

Dédicaces

J'exprime mes grandes reconnaissances et mes vifs remerciements à mon encadreur au CERIST M^{me}"Bessai F.Z et mon promoteur Mlle BOUSTIA .N à USDB pour le temps et l'attention qu'il a bien voulu consacrer au bon déroulement de ce projet.

A mes très chère Parents
Vous êtes l'exemple de dévouement qui n'a pas cessé de m'encourager et de prier pour moi.
Rien au monde ne vaut les efforts fournis jour et nuit pour mon éducation et mon bien être. Ce travail est le fruit de vos sacrifices que vous avez consentis pour mon éducation et ma formation.
Puisse Dieu, le tout puissant, vous préserver et vous accorder santé, longue vie et bonheur.

A ma sœurs; Sarah que dieu la garde.

A mes oncles, tantes, cousin et cousines affectueuses reconnaissances.

A mes enseignants de l'école primaire jusqu'à l'université dont les conseils précieux m'ont guidée qu'ils trouvent ici l'expression de ma reconnaissance.

A mon binôme Ramzy qui a partagé avec moi les moments difficiles de ce travail et à sa famille.

A mes amis; Oussama, Nadjib, Adel, Amine, Ishak, Mohamed, Billel, Walid, Sofiane, et mes amis de Dar EL Gharnatia et je remercie aussi tous ceux qui de près ou de loin ont bien voulu m'encourager pour que ce travail puisse être achevé.

Je vous remercie de votre patience vous m'avais aidé toujours à avancer vous êtes tous des grands amis très gentilles, merci d'être toujours près de moi, amis avec lesquelles je souris.

Je tiens à remercier ALLAH le tout puissant de m'avoir donné l'énergie et la force d'accomplir ce travail.

Khalil

Dédicaces

Au terme de ce travail, je tiens à exprimer ma profonde gratitude et mes sincères Remerciements à mes tuteurs de stage au CERIST M^{me}"Bessai F.Z et Mlle BOUSTIA .N à USDB.
Pour tout le temps qu'ils m'ont consacré, leurs directives précieuses, et pour la qualité de leur suivi durant toute la période de mon stage.

A l'âme de mon père Ayadi Abderrahmane qui m'a quitté sans voir le fruit de son éducation. Lui qui m'a transmis l'amour de vivre, l'amour de sacrifice et celui de continuer à donner sans limite.

A ma très cher mère Lakehal Zahia, Autant de phrases aussi expressives soient-elles ne sauraient montrer le degré d'amour et d'affection que j'éprouve pour toi. Tu m'as comblé avec ta tendresse et affection tout au long de mon parcours. Puisse le tout puissant te donner santé, bonheur et longue vie afin que je puisse te combler à mon tour.

A mes deux petites sœurs « Camélia et Célia » que dieu les garde.

Mes remerciements aux membres de mon groupe musical « BABYLONE », ainsi qu'à tous mes amies ils se reconnaitront.

Mes remerciements vont à toute personne qui a contribué de près ou de loin à l'élaboration de ce travail.

Je tiens à la fin de ce travail à remercier ALLAH le tout puissant de m'avoir donné la foi et de m'avoir permis d'en arriver là.

Ayadi Ramzy.

Liste des tableaux

Liste des figures

Table des matières

Chapitre 3 : La recherche d'information agrégée

Chapitre 4 : Proposition pour l'agrégation des résultats dans la recherche d'information structurée

Chapitre 5 : Implémentation du système

Introduction général

Introduction Générale

Les Systèmes de Recherche d'Information (SRI) renvoient en réponse à une requête une liste de documents potentiellement pertinents. L'information pertinente recherchée peut se retrouver entièrement dans un document ou être éparpillée dans plusieurs documents (Boughanem, Savoy, 2008). L'utilisateur doit alors parcourir la liste des documents retournés et regrouper et sélectionner les parties (ou granules d'information) qu'il juge pertinentes. Un granule d'information peut être une définition, une image, un texte descriptif, une vidéo, un tableau, ou même un attribut avec sa valeur (par exemple une adresse, un téléphone, etc). La réponse pertinente idéale serait donc composée de tous les granules sélectionnés par l'utilisateur.

Plutôt que de laisser l'utilisateur construire lui-même son résultat idéal, une alternative serait d'extraire et de regrouper automatiquement les granules d'information nécessaires. Ce genre d'approche est ce que l'on appelle la recherche agrégée. L'agrégation d'information à partir de plusieurs sources peut aider à satisfaire de nombreux besoins d'information. Par exemple, pour les requêtes de type instance ("Alger") ou classe ("ville d'Algérie"), qui représentent 71% des requêtes posées sur le Web d'après (Kato et al., 2009), la liste n'est vraiment ni la meilleure ni la plus naturelle des réponses.

Ainsi, pour la requête "Restaurant à Alger", retourner tous les documents qui contiennent les termes "restaurant" "Libanais" et "Blida" risque de couper l'appétit à l'utilisateur. La construction d'un résultat agrégé semble être une meilleure solution.

Notre travail se situe dans le contexte de la *recherche d'information*, plus précisément dans la recherche d'information dans des documents structurés (XML).

Chapitre 1 : Concepts de base de la Recherche d'Information

1.1. Introduction

La recherche d'information est une branche de l'informatique qui s'intéresse à l'acquisition, l'organisation, le stockage, la recherche et la sélection d'information répondant aux besoins des utilisateurs **(Salton, 1970) (Salton, 1984)**. Un Système de Recherche d'Information *(SRI)* est un ensemble de programmes informatiques qui a pour but de sélectionner des informations pertinentes répondant à des besoins utilisateurs, exprimés sous forme de requêtes. Trois notions clés caractérisent un SRI : document, requête et pertinence.

Un document désigne toute unité qui peut présenter une réponse à une requête donnée. En effet, un document peut être un morceau de texte, une page Web, une image, une séquence vidéo, etc. En outre, les documents textuels peuvent avoir plusieurs spécifications ; un document peut être un texte sans aucune structuration ou bien des documents semi-structurés de type XML par exemple. Les documents peuvent aussi être complètement structurés comme par exemple les formulaires.

Une requête exprime le besoin d'information d'un utilisateur. Elle peut être exprimée selon différents langages. Le langage le plus utilisé est le langage naturel.

La pertinence est une notion fondamentale en *RI*. Elle est l'objet de tout système de recherche d'information. Elle peut être définie comme la correspondance entre un document et une requête selon le système ou l'utilisateur.

Dans ce chapitre, nous commençons à présenter brièvement le processus de *RI* traditionnelle, ce processus cherche à mettre en correspondance une collection de documents et une requête utilisateur à travers un système de recherche d'information, composé d'un module d'indexation et d'un module d'appariement document-requête.

Nous passerons ensuite en revue les principales modèles utilisés pour l'appariement entre la requête et les documents. Ces modèles sont étroitement liés, mais on distingue trois principaux courants : les modèles basés sur la théorie des ensembles, issus du modèle booléen, les modèles algébriques, dont le premier représentant a été le modèle vectoriel et les modèles probabilistes.

Enfin, nous présenterons les diverses mesures et collections utilisées pour évaluer ces modèles et systèmes.

ponctuations, etc.

1.2.3.2. L'élimination des mots vides

Un des problèmes majeurs de l'indexation consiste à extraire les termes significatifs des mots vides (pronoms personnels, prépositions, ...). Les mots vides peuvent aussi être des mots athématiques (les mots qui peuvent se retrouver dans n'importe quel document parce qu'ils exposent le sujet mais ne le traitent pas, comme par exemple contenir, appartenir. Les mots vides sont des mots peu significatifs augmentant ainsi la taille de l'index et rendant la recherche plus lente. De ce fait l'élimination est une étape indispensable. On distingue deux techniques pour éliminer les mots vides:

- L'utilisation d'une liste de mots vides (aussi appelée anti-dictionnaire),
- L'élimination des mots dépassant un certain nombre d'occurrences dans la collection.

1.2.3.3. Lemmatisation

Un mot donné peut avoir différentes formes dans un texte. On peut par exemple citer économie, économiquement, économétrie, économétrique, etc. Il n'est pas forcement nécessaire d'indexer tous ces mots et un seul suffirait à représenter le concept véhiculé. Pour résoudre le problème, une substitution des termes par leur racine ou lemme est utilisée. Plusieurs méthodes de lemmatisation ont été proposés dans la littérature, parmi lesquelles : la troncature, la méthode des n-grammes **(Adamson et Boreham, 1974)**, les dictionnaires ou l'élimination des affixes (algorithme de Porter **(Porter, 1980)**).

1.2.3.4. Pondération des termes

La pondération permet d'assigner aux termes leur degré d'importance dans les documents. Un terme peut être expressif s'il apparaît suffisamment fréquemment pour être statistiquement important sans toutefois excéder une certaine limite qui le classerait dans la catégorie des mots outils (vides). La plupart des techniques de pondération sont basées sur deux facteurs : la pondération locale qui quantifie la représentativité locale d'un terme dans le document (*tf : TermFrequency*), et la pondération globale qui quantifie la représentation du terme vis-à-vis de la collection complète des documents (*idf : Inverse of Document Frequency*) :

- ***tf (TermFrequency)*** **:** cette mesure est proportionnelle à la fréquence du terme dans le document. Plusieurs formules de pondérations ont été proposées, parmi lesquelles : la fonction brute (nombre d'occurrences), ou bien combiné avec d'autre fonction (exemple : *log (tf)*,...)

- ***idf (Inverse of Document Frequency)*** **:** ce facteur mesure l'importance d'un terme dans toute la collection. Un terme qui apparaît souvent dans la base documentaire ne doit pas avoir le même impact qu'un terme moins fréquent. Il est généralement exprimé comme suit : *log(N/df)*, où *df* est le nombre de documents contenant le terme et *N* est le nombre total de documents de la base documentaire.

La mesure *(tf * idf)* donne une bonne approximation de l'importance du terme dans le document, particulièrement dans les corpus de documents de taille homogène.

1.3. Appariement document-requête

L'appariement entre le document et la requête permet de calculer une mesure appelée pertinence système, supposée représenter la pertinence du document vis-à-vis de la requête, elle est vue comme une probabilité ou une similarité vectorielle, notée : *RSV (Q,d) (RetrievalStatus Value)*, où *Q* est une requête et *d* un document. Cette mesure tient compte du poids des termes dans les documents. L'appariement document-requête et le modèle d'indexation permettent de caractériser et d'identifier un modèle de recherche d'information.

Un modèle de RI a pour objectif de fournir une formalisation du processus de recherche d'information. Il doit accomplir le plus important rôle qui est de fournit un cadre théorique pour la modélisation de la mesure de pertinence.

De nombreux modèles de recherche ont été proposés dans la littérature. Nous distinguons trois principaux modèles : les modèles ensemblistes, algébriques et probabilistes, présentés dans la figure suivante :

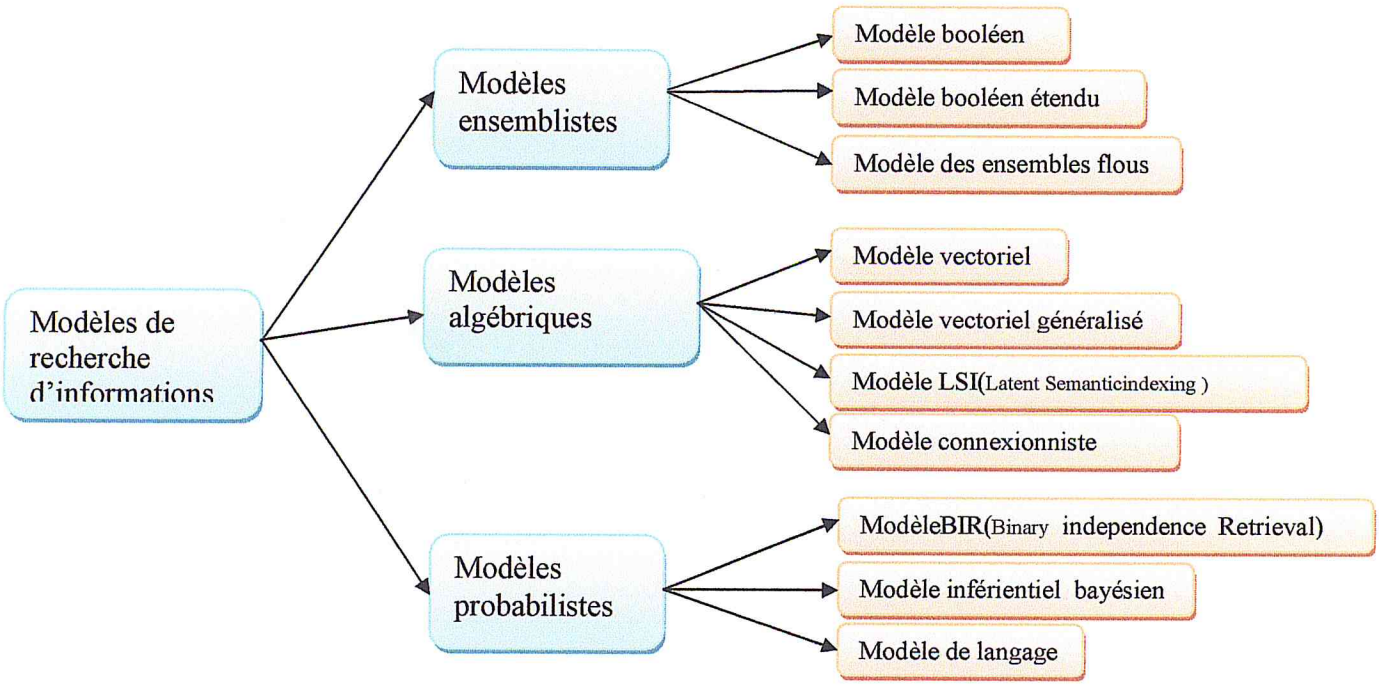


Fig. 1.2: Taxonomie des modèles de recherche d'information

Les modèles ensemblistes reposent sur la théorie des ensembles, où les termes de la requête sont séparés par des opérateurs logiques : conjonction (ET), disjonction (OU) et négation (NON). Ces opérateurs permettent d'effectuer des opérations d'union, d'intersection et de différence entre les ensembles de résultats associés à chaque terme.

Les modèles algébriques se basent sur la théorie algébrique, où la pertinence d'un document vis-à-vis d'une requête est définie par des mesures de distance dans un espace vectoriel.

Enfin, les modèles probabilistes se basent sur la théorie des probabilités, où la pertinence d'un document vis-à-vis d'une requête est vue comme une probabilité de pertinence document/requête.

Dans les sections suivantes, nous décrivons pour chaque type de modèle, le modèle le plus représentatif (à savoir : le modèle booléen, le modèle vectoriel et le modèle probabiliste)

1.3.1. Le modèle booléen

Le modèle booléen est historiquement le premier modèle de RI, Il a été introduit en 1983 par Salt on et McGill.

Ce modèle est basé sur la manipulation des ensembles et l'algèbre de Boole. Un document est représenté par une liste de termes (termes d'indexation). Une requête est représentée sous forme d'une équation logique. Les termes d'indexation sont reliés par des connecteurs logiques ET, OU et NON.

Le processus de recherche mis en œuvre consiste à effectuer des opérations sur l'ensemble de documents afin de réaliser un appariement exact avec l'équation de la requête. L'appariement exact est basé sur la présence ou l'absence des termes de la requête dans les documents. La similarité entre un document et une requête est définie par :

$$\begin{cases} rsv(Q,d) = 1 & si\ d\ appartient\ à\ l'ensemble\ décrit\ par\ la\ requête \\ 0 & Sinon \end{cases}$$

La décision binaire sur laquelle est basée la sélection d'un document ne permet pas d'ordonner les documents renvoyés à l'utilisateur selon un degré de pertinence.

1.3.2. Le modèle vectoriel

C'est un modèle qui préconise la représentation des requêtes utilisateurs et des documents sous forme de vecteurs, dans l'espace engendré par tous les termes d'indexation. D'une manière formelle, les documents (d_j) et les requêtes Q sont des vecteurs dans un espace vectoriel des termes d'indexation (t_1, t_2, …, t_T) de dimension T et représentés comme suit :

$$\overrightarrow{d_j} = [d_{j1}, d_{j2}, \ldots, d_{jT}] , = [\vec{q}_1, q_2, \ldots, q_T]$$

Où d_{ji} et q_i sont respectivement les poids des termes t_i dans le document d_j et la requête Q. D'après ce modèle, le degré de pertinence d'un document relativement à une requête est perçu comme le degré de corrélation entre les vecteurs associés. Ceci nécessite alors la spécification d'une fonction de calcul de similarité entre vecteurs mais également d'une fonction de pondération des termes. La plus répandue est celle de Sparck et Needham**(Spark et al., 1972)** qui définit le poids d'un terme t_i dans un document d_j comme suit :$d_{ji} = tf_{ji} * idf_i$

Où :

tf_{ji}: est la fréquence relative du terme t_i dans le document d_j.

idf_i: est l'inverse de la fréquence absolue du terme t_i dans la collection.

$idf_i = \log\frac{N}{n_i}$; avec n_i le nombre de documents contenant le terme t_i et N est le nombre total de documents dans la collection.

La fonction de similarité permet de mesurer la ressemblance des documents et de la requête. Une des plus simple est celle du produit scalaire développé par *Gérard Salton* et son équipe **(Salton, 1970)** dans leur projet SMART (Salton's Magical Automatic Retriever of Text) comme suit :

$$rsv(\vec{d_j},\vec{Q}) = \sum_{i=1}^{t} w_{ij} \times w_{iq}$$

Parmi les fonctions de similarité les plus répandues dans la littérature, on peut citer les mesures Cosinus [3], Jaccard et Dice :

Mesure de cosinus :

$$Sim(\vec{d_j},\vec{Q_k}) = \frac{\sum_{i=1}^{N}(wd_{ij} \times wq_{ik})}{\sqrt{\sum_{i=1}^{N} wd_{ij}^{\,2} \times \sum_{j=1}^{N} wq_{ik}^{\,2}}}$$

Mesure de Jaccard :

$$Sim(\vec{d_j},\vec{Q_k}) = \frac{\sum_{i=1}^{N}(wd_{ij} \times wq_{ik})}{\sum_{i=1}^{N} wd_{ij}^{\,2} + \sum_{i=1}^{N} wq_{ik}^{\,2} - \sum_{i=1}^{N}(wd_{ij} \times wq_{ik})}$$

Les avantages du modèle vectoriel sont nombreux : il permet la pondération des termes, il permet de renvoyer des documents qui répondent approximativement à la requête et les documents sont restitués dans un ordre décroissant de leur degré de similarité avec la requête. Plus le degré de similarité d'un document est élevé, plus le document ressemble à la requête et

plus il est susceptible d'être pertinent pour l'utilisateur.

Le modèle vectoriel suppose l'indépendance entre termes. En effet, la représentation vectorielle considère chaque terme séparément alors qu'on peut avoir des termes qui sont en relation sémantique entre eux.

1.3.3. Le modèle probabiliste

Le modèle probabiliste aborde le problème de la recherche d'information dans un cadre probabiliste. Le premier modèle probabiliste a été proposé au début des années 1960. Le principe de base consiste à présenter les résultats de recherche d'un SRI dans un ordre basé sur la probabilité de pertinence d'un document vis-à-vis d'une requête. Robertson **(Robertson, 1977)** résume ce critère d'ordre par le "principe de classement probabiliste", aussi désigné par *PRP* (*Probability Ranking Principle*).

Pour mesurer cette pertinence le modèle probabiliste se base sur la distribution des termes dans un échantillon représentatif de documents d'apprentissage. Les hypothèses posées sont les suivantes :

- la distribution des termes dans les documents pertinents est la même que leur distribution par rapport à la totalité des documents ;

- les variables : « document pertinent », « document non pertinent » sont indépendantes. Le processus de recherche se traduit par le calcul du degré (ou probabilité) de pertinence d'un document vis-à-vis à une requête.

Deux probabilités conditionnelles sont utilisées dans le processus de décision :

- $P(w_{ij}/ Pert)$: probabilité que le terme t_i occurre dans le document d_j sachant que ce dernier est pertinent pour la requête.

- $P(w_{ij}/ NonPert)$: probabilité que le terme t_i occurre dans le document d_j sachant que ce dernier n'est pas pertinent pour la requête.

Plusieurs solutions ont été proposées pour représenter le document D et pour estimer les paramètres du modèle. Parmi elles le modèle 2-poisson **(Robertson et al., 1994)$# (Walker et al., 2006)**. Le résultat de ces travaux est la fameuse formule*BM25*. La formule est la suivante :

$$rsv(Q,d) = \sum_{t \in Q} \frac{qt_f \times (k_2+1)}{k_2 \times qt_f} \times \frac{tf_{ij} \times (k_1+1) \times \log \frac{N - n_i + 0.5}{n_i + 0.5}}{k_1 \times ((1-b) + b \frac{l_{d_j}}{avg\_dl}) + tf_{ij}}$$

Avec :

qtf: la fréquence du terme t dans la requête, $_{dj}$: la longueur du document d_j; $l_{dj}= \sum_{i \in d_j} tf_{ij}$

tf_{ij}, les auteurs ont aussi proposé de mesurer en octets les longueurs des documents ;avg_dl: la longueur moyenne des documents de la collection. Elle est calculée comme

suit : avg_dl =, $\sum_{j \in N} \sum_{i \in T} \frac{tf_{ij}}{N}$

N : le nombre de documents de la collection ;

n_i: le nombre de documents contenant le terme t_i, T le nombre de termes de la collection.

k_1, $k_2$et b sont des constantes.

Les expérimentations ont montré que $k_1= 1.2$, $k_2= 0.8$, $b = 0.75$ ont donné les meilleurs résultats, en termes de performances

1.4. Conclusion

Dans ce chapitre, nous avons passé en revue les méthodes et modèles fondamentaux utilisés en recherche d'information « traditionnelle ».

Les SRI que nous avons décrits ici fonctionnent généralement sur des documents multi-formats (structurés, non structurés, balisés ou non balisés). Ils n'exploitent cependant que le contenu sémantique des documents, c'est à dire leur texte.

Devant le nombre croissant de documents semi-structurés ou structurés mis à disposition, de nouveaux modèles cherchent à tirer parti de l'information structurelle très dense contenue dans ce type de documents, en combinant cette dernière avec l'information de contenu. Dans le chapitre suivant, nous nous intéressons particulièrement à ces nouveaux modèles pour la recherche d'information structurée. Les principales problématiques qu'ils cherchent à résoudre (interrogation, indexation, identification des éléments pertinents) sont comparables à celles soulevées dans la RI traditionnelle, mais doivent être abordées en ajoutant la dimension structurelle à la dimension de contenu.

CHAPITRE 2

Document Structurés et enjeux de la Recherche d'Information Structurée

Chapitre 2 : Documents Structurés et enjeux de la Recherche d'Information Structurée

2.1. Introduction

Le type des documents mis à disposition des utilisateurs évolue : du simple document texte « plat », on assiste aujourd'hui à la généralisation des documents structurés. Des formats tels que SGML (Standard Generalized Markup Language) ou encore XML (eXtensible Markup Language), conçus à l'origine pour faciliter l'échange et la standardisation des données, voient leur importance augmenter grâce à l'expansion d'Internet (Mignet et al., 2003). Ce chapitre a pour objectif de présenter les différentes problématiques soulevées par la RI structurée, ainsi que les différentes solutions proposées dans la littérature. Nous commençons tout d'abord par définir la notion de structure et présenter les documents structurés. Nous présentons ensuite les différents défis soulevés par la recherche d'information dans les documents structurés. Ce chapitre nous permet aussi de distinguer les grands types d'approches proposées dans la littérature, à savoir les approches orientées BD et les approches orientées RI.

Afin d'utiliser au mieux les propriétés des documents structurés, de nouvelles techniques d'indexation, ainsi que de nouveaux langages d'interrogation prenant en compte la structure sont utilisés.

2.2. Documents structurés

Les documents structurés (Sévigny, 2002) sont des documents qui contiennent de l'information à propos de leur structure logique, sémantique et intellectuelle. Pour mieux comprendre, voici un exemple d'un document non structuré suivi du même document mais cette fois-ci structuré:

Document non structuré

```
<?xml version="1.0"?>
<document>
<p><font size="20pt"><b>Introduction</b></font></p>          /Non structuré /
<p><i>George Washington</i> n'a jamais gouverné le
<b>Washington</b> mais a résidé à
<b>Washington</b>.</p>
</document>
```

Document structuré

```
<?xml version="1.0"?>
<document>
<section>
<titre>Introduction</titre>
<p><personne>George Washington</personne> n'a jamais                    /structuré /
gouverné le <etat>Washington</etat> mais a résidé à
<ville>Washington</ville>.</p>
</section>
</document>
```

La différence entre ces documents est très grande. Dans le premier cas, il s'agit d'un document formaté pour une application particulière. Dans le deuxième cas, il s'agit d'une structure d'information, et c'est tout.

Supposons que nous bâtissions deux collections de documents semblables au précédent. Si nous créons les documents selon le premier modèle, nous pourrions obtenir un système d'information intéressant, mais limité. Pour voir ces limites, essayons de voir ce que l'on peut faire avec la deuxième forme:

On peut afficher les documents en plusieurs formats

On peut créer un index des personnes, des états, des villes

On peut construire une table des matières automatiquement

Ces opérations sont permises parce que le deuxième document, contrairement au premier, ne contient pas d'informations sur les traitements, mais bien sur la nature des informations. Et il s'agit de l'intérêt principal des documents structurés.

2.2.1. Les Documents XML

XML (entendez eXtensible Markup Language) est en quelque sorte un langage HTML amélioré permettant de définir de nouvelles balises. Il s'agit effectivement d'un langage permettant de mettre en forme des documents grâce à des balises (markup). Contrairement à HTML, qui est à considérer comme un langage défini et figé (avec un nombre de balises limité), XML peut être considéré comme un métalangage permettant de définir d'autres langages, c'est-à-dire définir de nouvelles balises permettant de décrire la présentation d'un texte (Qui n'a jamais désiré une balise qui n'existait pas ?).

XML a été mis au point par le XML Working Group sous l'égide du World Wide Web Consortium (W3C) dès 1996. Depuis le 10 février 1998, les spécifications XML 1.0 ont été reconnues comme recommandations par le W3C, ce qui en fait un langage reconnu. (Tous les documents liés à la norme XML sont consultables et téléchargeables sur le site web du W3C, http://www.w3.org/XML/)

XML est un sous ensemble de SGML (Standard Generalized Markup Language), défini par le standard ISO8879 en 1986, utilisé dans le milieu de la Gestion Electronique Documentaire (GED). XML reprend la majeure partie des fonctionnalités de SGML, il s'agit donc d'une simplification de SGML afin de le rendre utilisable sur le web !

2.2.2 La syntaxe des documents XML

La syntaxe des documents XML est définie par des balises encadrant les portions d'informations. Une balise (ou tag ou label) est une suite de caractères encadrés par "<" et ">", comme par exemple <nom_balise>.

Un élément est une unité sémantique identifiée, délimitée par des balises de début <A> et de fin </A>, comme par exemple : <ma_balise> texte </ma_balise>. Les éléments peuvent être imbriqués.

Les attributs des balises sont spécifiés au début de l'élément et après le nom de la balise, en utilisant la syntaxe nom_attribut=valeur_ attribut. Par exemple : <ma_balisenom_attribut = valeur_ attribut > texte </ma_balise>.

```
<?xml version="1.0"?>
<Document type = <<livre>>>
<titre>Introduction a L'XML</titre>
<Auteur>
<Nom>Benaouda</Nom>
<Prénom>Ramzy</Prénom>
</Auteur>
<Année>2012-2013 </Année>
</Document>
```

2.2.2.1. Structure des documents XML

En réalité 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. 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, XML fournit un moyen de vérifier la syntaxe d'un document grâce aux DTD (Document Type Définition). Il s'agit d'un fichier décrivant la structure des documents y faisant référence grâce à un langage adapté. Un document XML possédant une DTD et étant conforme à celle-ci est appelé document valide. Et enfin la dernière composante d'un fichier XML est l'arbre des éléments pourvue.

2.3. Enjeux de la Recherche d'Information Structurée

Le but des systèmes de recherche d'information est d'apporter une réponse non nécessairement exacte (au sens base de données) aux besoins en information de leurs utilisateurs.

En recherche d'information traditionnelle, les SRI, traitent les granules des collections (documents) dans leur globalité. Les notions de documents logiques et de document physique sont alors confondues, avec les documents structurés qui permettent le balisage des contenus des documents, on peut traiter l'information avec une granularité plus fine. Le but des SRI dans le contexte des documents structurés est alors de renvoyer des unités d'information auto-explicatives à l'utilisateur, et non des documents complets.

Dans des corpus de documents XML, chercher les nœuds les plus exhaustifs et spécifiques pour une requête revient donc à trouver les sous-arbres de taille minimale pertinents à la requête.

Afin de permettre ces différentes recherches, les techniques de la recherche d'information traditionnelle doivent être adaptées et de nouvelles méthodes doivent être proposées pour l'indexation, l'interrogation ou encore la recherche et le tri des unités d'information.

La dimension structurelle des documents structurés a soulevé plusieurs problématiques relatives à chaque phase du processus de recherche.

La problématique dans le cadre de l'indexation se situe essentiellement au niveau de l'information structurelle. Dans les SRI traditionnels, le contenu textuel est traité afin de trouver les termes les plus représentatifs des documents. Dans le cas des documents structurés, la dimension structurelle s'ajoute au contenu, et les questions suivantes se posent alors (Sauvagnat, 2005):

- quelle unité doit-on indexer de la structure des documents ?
- comment relier cette structure au contenu même du document ?
- en fonction de quelle dimension (niveau élément, documents, collection) doit-on pondérer les termes d'indexation ?

Considérons à présent l'interrogation des documents. Il s'agit ici de permettre à l'utilisateur la possibilité d'exprimer des besoins diversifiés (concernant le contenu des documents et/ou la structure), et ce de manière simple.

La dernière problématique est celle des modèles de recherche et de tri des unités d'informations. Les systèmes de recherche doivent pouvoir décider de la granularité de l'information à renvoyer

s'il s'agit d'une requête orientée contenu seulement. S'il s'agit d'une requête orientée contenu et structure deux cas sont envisageables :
- l'utilisateur spécifie le type d'éléments à renvoyer ;
- l'utilisateur ne spécifie pas le type d'éléments à renvoyer.
Dans le deuxième cas, c'est au système de décider de la granularité de l'information à renvoyer.
Ce sont trois problématiques spécifiques à la RI structurée et elles restent toujours d'actualité, vue la jeunesse de ce domaine d'intérêt.

2.4. Indexation et langages de requêtes

2.4.1. Indexation de documents structurés

L'indexation permet de représenter les documents de manière à faciliter la recherche et de la rendre plus efficace. L'indexation des documents structurés diffère de celle utilisée pour des documents textes « plats » par le fait de la prise en compte de la dimension structurelle qui s'ajoute au contenu. De ce fait, un schéma d'indexation de document XML devrait principalement permettre la reconstruction du document XML décomposé dans des structures de stockage et la recherche par mot clé et par expressions de chemin sur la structure XML.

2.4.1.1. Indexation de l'information textuelle

L'indexation de l'information textuelle, c'est-a-dire l'extraction et la pondération des termes, est similaire à la RI classique. Sa spécificité dans les documents semi-structurés et notamment les documents XML, réside dans la description des relations entre les termes et l'information structurelle : c'est ce qu'on appelle la "portée des termes d'indexation".

-Portée des termes d'indexation

Pour relier les termes a l'information structurelle, deux solutions ont été proposées dans la littérature : les sous-arbres imbriques et les unités disjointes :

- **Sous-arbres imbriques** : On considère que le contenu de chaque nœud de l'index est une unité atomique. Les termes des nœuds feuilles sont donc propagés dans l'arbre des documents. Comme les documents XML possèdent une structure hiérarchique, les nœuds de l'index sont imbriqués les uns dans les autres et par conséquent, l'index contient des informations redondantes.

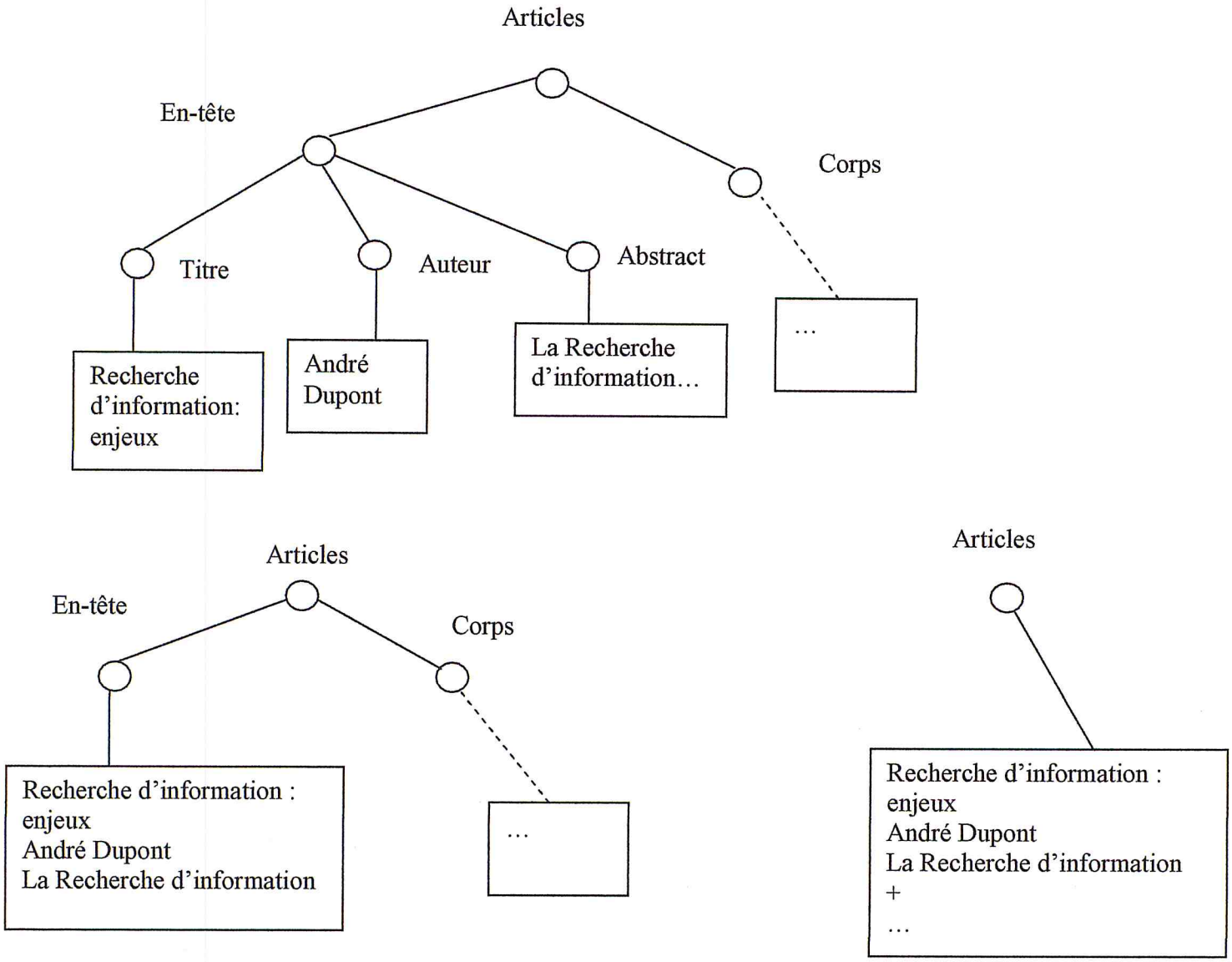


Fig. 2.1 : Indexation de sous arbres imbriqués **(Sauvagnat, 2005)**

-**Unité disjointes** : Dans ces approches, le document XML est décomposé en unités disjointes, de telle façon que le texte de chaque nœud de l'index est l'union d'une ou plusieurs parties disjointes. Les termes des nœuds feuilles sont uniquement reliés à un et un seul nœud. Une fois les unités d'indexation spécifiées, il reste à pondérer les termes. Cette tâche est une adaptation des fonctions de pondération déjà proposées en RI classique.

Si on reprend en exemple l'arbre de la figure 2.2, les termes « recherche d'information enjeux » seront uniquement reliés au nœud /article/en-tête/titre, les termes « André Dupond » au nœud /article/entête/auteur et les termes « la recherche d'information » au nœud /article/en-tête/abstract. Le nœud /article/entête n'est quant à lui relié à aucun terme.

Différentes approches ont été proposées dans la littérature pour indexer l'information structurelle selon des granularités variées. On distingue trois types d'approches pour l'indexation de l'information structurelle :

- Indexation basée sur des champs

Dans cette méthode d'indexation, le document est représenté comme un ensemble de champs et du contenu associé à chaque champ. Plusieurs façons permettent d'obtenir les différents champs d'un document XML :

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

- Ils peuvent être codés en tant que métadonnées dans les fichiers XML, par exemple en utilisant RDF.
- Ils sont simplement extraits de la DTD ou du schéma XML associé.

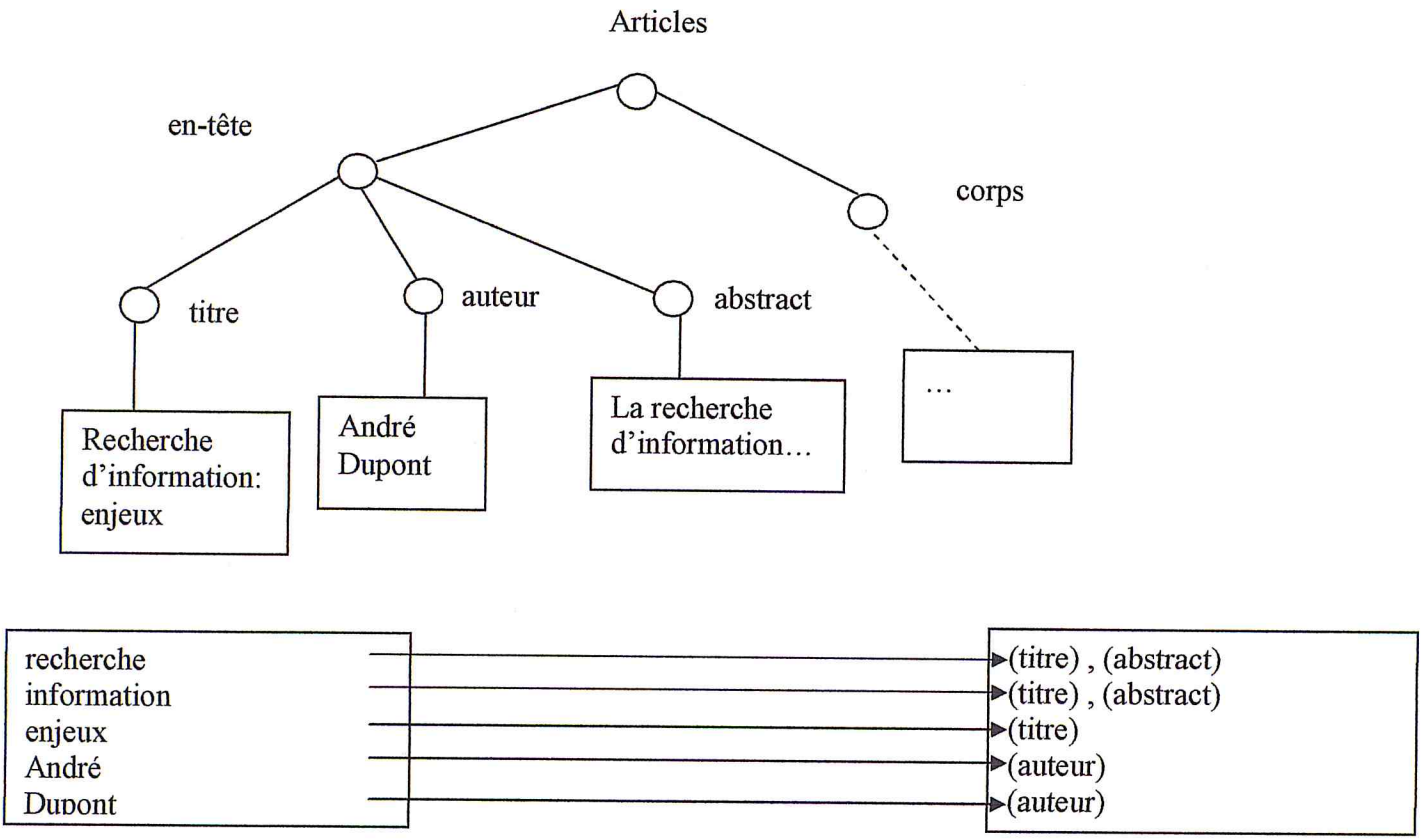


Fig.2.2: Exemple d'indexation basée sur des champs **(Sauvagnat, 2005)**

Indexation basée sur des chemins

Ce type de techniques facilite la navigation dans les documents en permettant la résolution des expressions XPath, il permet aussi de retrouver des documents ayant des valeurs connues pour certains éléments ou attributs.
Ces techniques souffrent de la difficulté de retrouver les relations ancêtre-descendant entre les différents nœuds des documents. La figure suivante (figure 2.3) représente une illustration de ce type d'indexation.

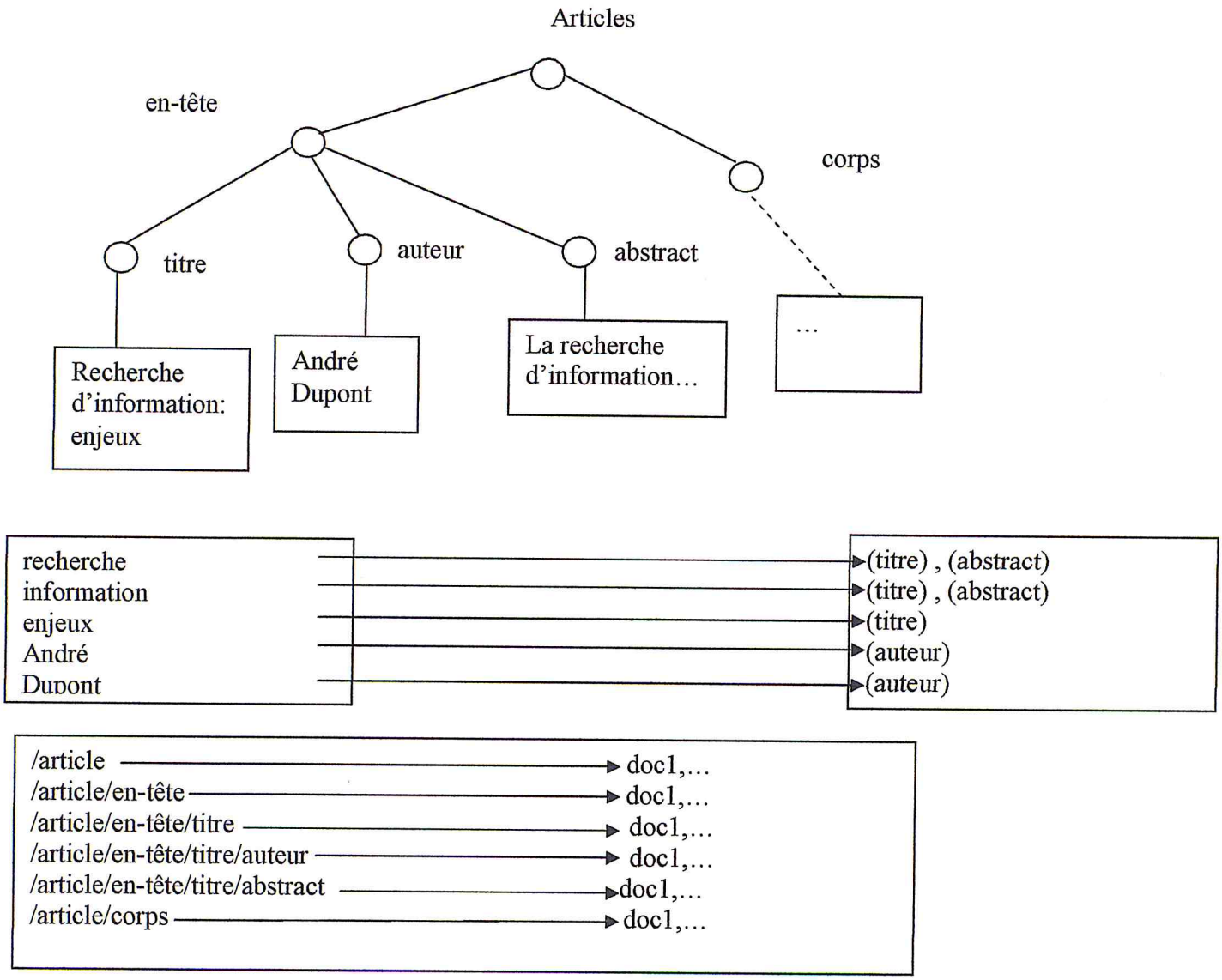


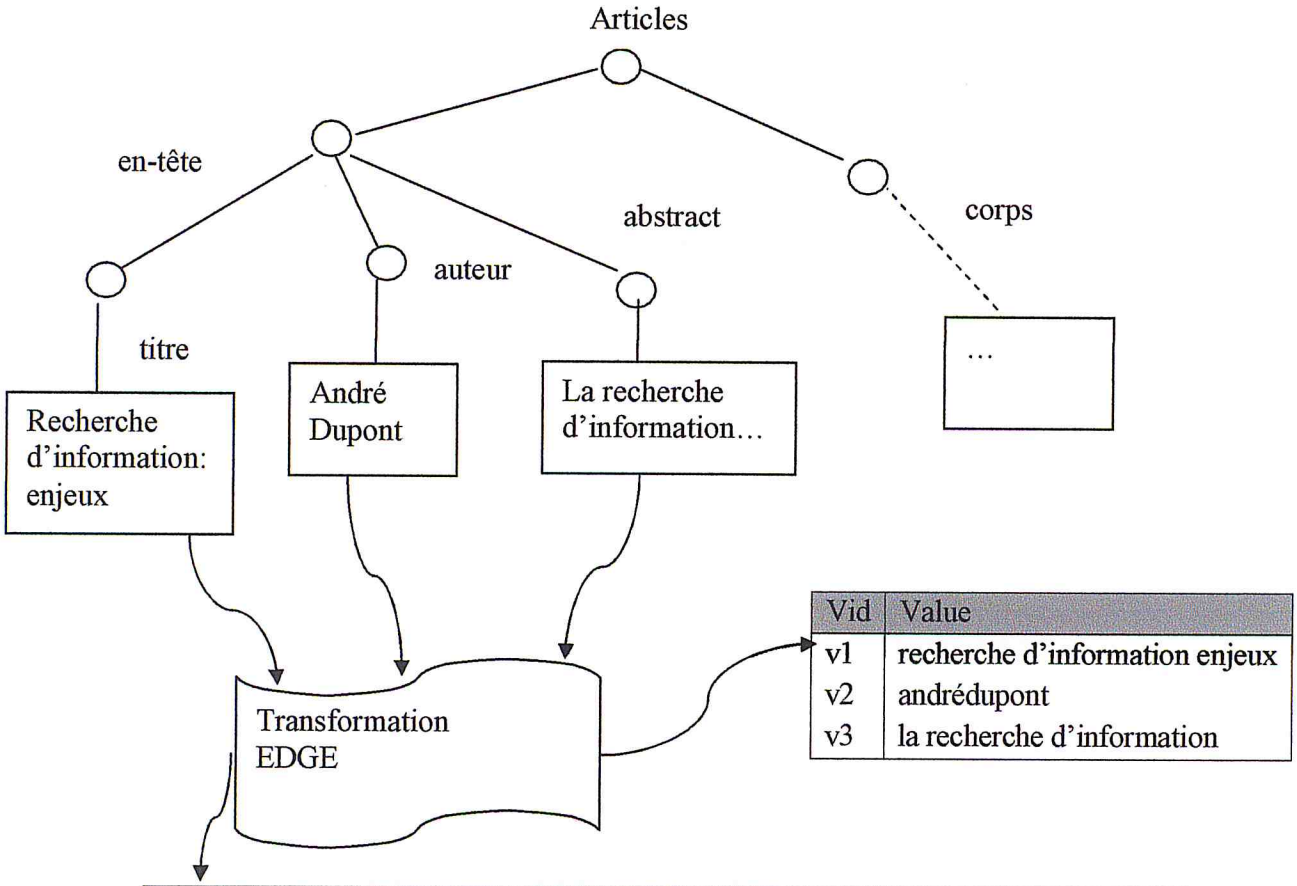
Fig.2.3: Exemple d'indexation basée sur des chemins **(Sauvagnat, 2005)**

-Indexation basée sur des arbres

Les nœuds d'un arbre sont numérotes dans l'index de façon à pouvoir reconstruire la structure arborescente des documents. Cette approche a été adaptée dans plusieurs systèmes de recherche, parmi lesquels citons les systèmes : EDGE et BINARY.

Dans le système EDGE (Florescu, 1999), une table appelée EDGE stocke la structure spécifique des documents XML, c'est à dire les arcs de la représentation en arbre des documents. Cette table, comme le montre la figure 2.4, stocke ainsi l'identifiant du nœud source et cible de chaque arc, l'ordre d'apparition des nœuds, le

nom du nœud cible et le type du nœud cible (interne ou feuille). Si le nœud cible est un nœud feuille, une table séparée stocke la valeur du nœud. L'inconvénient principal de l'approche EDGE est que de nombreuses requêtes ont des performances médiocres car elles nécessitent de nombreuses jointures sur la grande table EDGE.

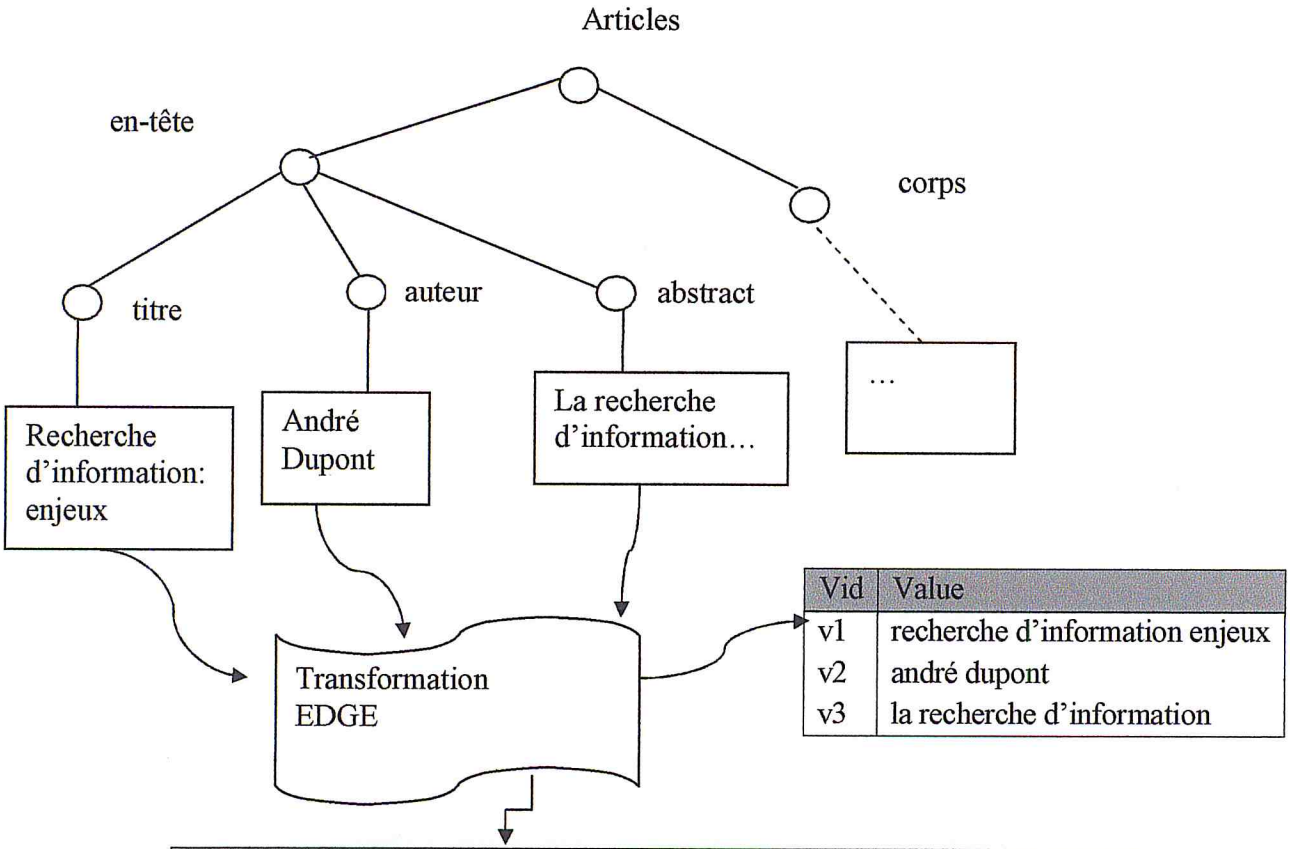


Vid	Value
v1	recherche d'information enjeux
v2	andrédupont
v3	la recherche d'information

Source	ordinal	name	datatype	target
NULL	1	article	ref	1
1	1	en-tête	ref	2
2	1	titre	ref	3
3	1	#titre	string	v1
2	2	auteur	ref	4
4	1	#titre	string	v2
2	3	abstract	ref	5
5	1	#titre	string	v3
1	2	corps	ref	6

Fig. 2.4 : *Transformation d'un document XML avec l'approche EDGE*(**Sauvagnat, 2005**)

De manière similaire à EDGE, l'approche BINARY (Florescu, 1999) matérialise la structure en arbre des documents XML dans des tables. L'approche BINARY crée une table séparée Bname pour chaque élément name. En d'autres termes, BINARY réalise une partition horizontale de la table EDGE en utilisant le nom de l'élément comme critère de partition (voir figure 2.5).



Vid	Value
v1	recherche d'information enjeux
v2	andré dupont
v3	la recherche d'information

$B_{article}$

source	ordinal	name	datatype	target
NULL	1	article	ref	1

$B_{en\text{-}tête}$

source	ordinal	name	datatype	target
1	1	en-tête	ref	2

B_{titre}

source	ordinal	name	datatype	target
2	1	titre	ref	3

source	ordinal	name	datatype	target
2	2	auteur	ref	4

B_{auteur}

source	ordinal	name	datatype	target
2	3	abstract	ref	5

$B_{abstract}$

source	ordinal	name	datatype	target
1	2	corps	ref	6

B_{corps}

source	ordinal	name	datatype	target
3	1	#titre	string	v1
4	1	#titre	string	v2
5	1	#titre	string	v3

Fig. 2.5 : *Transformation d'un document XML avec l'approche BINARY* **(Sauvagnat, 2005)**

Par comparaison à l'approche EDGE, les tables ont une taille moins importante, mais le nombre de jointures nécessaires pour des requêtes de chemin est toujours aussi grand. L'approche BINARY est cependant très efficace pour des recherches sur un élément particulier.

2.4.2. Langages de requêtes

Les utilisateurs devraient pouvoir exprimer leurs besoins selon deux catégories de requêtes :

1. des requêtes composées de simples mots clés comme en RI. C'est le cas lorsque les utilisateurs n'ont pas d'idée précise de ce qu'ils recherchent ou n'ont pas de connaissance concernant la structure des documents.
2. des requêtes composées de contraintes sur le contenu (donc de mots clés) et de contraintes structurelles. La majorité des langages de requêtes proposés dans la littérature sont issus de la communauté des bases de données. D'une manière générale, les langages de requêtes doivent supporter à la fois des contraintes portant sur le contenu et sur la structure. Il existe de nombreux langages d'interrogation. Parmi eux citons XQuery, XQL, NEXI, XOR, XML-QL, Quilt, XML-GL, XIRQL, Tequyla-TX ou Tex-Query. Notons simplement que nombreuses sont les spécifications de langages mais rares sont les implémentations.

2.5. Conclusion

Les documents structurés, en permettant le balisage des contenus des documents, réactualisent la problématique de recherche d'information classique, et permettent ainsi de traiter l'information avec une granularité plus fine. Le but des SRI traitant des documents structurés est alors d'identifier des parties des documents les plus pertinentes à une requête donnée.

La majorité des modèles proposés aujourd'hui tente de trouver, comme réponse à la requête de l'utilisateur, l'élément (le sous arbre) le plus pertinent répondant à la requête. Mais la vision de chercher l'élément (le sous arbre) le plus pertinent répondant à la requête, à une limite très forte, car l'information pertinente peut ne pas être un seul élément dans le document, par contre elle peut être un ensemble d'éléments du même document et qui ne sont pas contigües. Pour ce type de problème la recherche d'information agrégée a été introduite récemment dans la littérature.

CHAPITRE 3

La Recherche d'information agrégée

Chapitre 3 : La recherche d'information agrégée

3.1. Introduction

La plupart des travaux en recherche d'information s'est concentrée sur la recherche des documents et la liste ordonnée de résultats est la réponse la plus utilisée (Rivadeneira et al., 2003). Cette présentation sous forme d'une liste ordonnée, utilisée par la majorité des SRI actuels, impose à l'utilisateur de la parcourir séquentiellement et d'examiner les réponses une par une afin de trouver le contenu approprié (Oren et al.,1998). Cela prend beaucoup de temps et ce n'est pas la solution idéale.

Imaginons une recherche Web concernant un hôtel, qui n'a pas un site web propre, le résultat idéal peut impliquer une description de l'hôtel, une adresse, l'information de contact et ainsi de suite. Cependant, toute cette information n'est pas nécessairement dans la même page. L'utilisateur doit considérer et aller « fouiller » dans différentes sources telles que les agences de voyages en ligne, les adresses, etc. pour fabriquer la réponse idéale, comportant toutes les informations dont l'utilisateur a besoin.

La Recherche d'Information (RI) agrégée vient pour répondre à ce type de problème. Cela change le paradigme de recherche actuelle, où on renvoie à l'utilisateur une liste ordonnée. La recherche agrégée propose d'assembler dans un seul document, toutes les informations pertinentes et non redondantes pouvant répondre au besoin de l'utilisateur. Nous appelons ce dernier un document agrégé ou résultat agrégé ou tout simplement un agrégat. Un document agrégé est le résultat d'assembler des informations à partir de plusieurs sources (Kopliku A. et al., 2009).

3.2. De l'information focalisée aux documents agrégés

3.2.1. Définition

Dans l'atelier 2008 d'ACM SIGIR sur la recherche agrégée (Murdock et al.,2008), la définition suivante a été donnée :

Définition : La recherche agrégée est la tâche de rechercher et de regrouper des informations issues de sources différentes et de les placer dans une même interface.

En fait, l'utilisateur pourrait être satisfait d'une partie d'un document ou de plusieurs parties. Parfois, son besoin d'informations pourrait être la composition de quelques sections de différents documents.

Les documents agrégés (agrégats) sont établis par assemblage des granules d'informations. Dans le cadre de l'information textuelle, un granule peut représenter un passage du texte. Dans le cas des documents structurés ce sont des nœuds (sous arbre/élément) délimités par des balises.

3.2.2. Structure d'agrégat

La recherche agrégée doit traiter l'organisation du contenu pertinent. Il n'est pas suffisant de collecter (trouver) les informations mais il faut également les organiser pour la visualisation finale.

La structure d'agrégat est définie comme toute l'information qui décrit le contenu, l'ordre de la visualisation, et les préférences dans la visualisation du document agrégé (Kopliku A. et al., 2009).

Les exemples suivants illustrent des structures partielles de certaines informations agrégées :

- une liste de 4 revues, 3 informations supplémentaires
- paragraphe A, paragraphe B, paragraphe C, avec l'ordre de visualisation : A, B, C

Dans de RI traditionnelle la réponse à une requête est une liste de documents. Dans la recherche agrégée il devrait être possible d'ajouter l'information de la structure d'agrégat à la requête. Quelqu'un pourrait demander des revues. Mais les études d'utilisateurs (Nielsen, 2003) indiquent qu'ils sont généralement paresseux. Ils n'emploient pas des options additionnelles. Néanmoins, ceci peut être très utile dans certains cas. Imaginez qu'un service de voyage sur Web qui recherche des hôtels et il est intéressé par les hôtels avec au moins 3 photos, 1 carte, l'adresse, le numéro de téléphone, le nombre d'étoiles, les revues, etc. Ce type de recherche pourrait donc être utile dans ce cas.

3.2.3 Les phases d'agrégation

Nous pouvons distinguer trois phases abstraites dans la recherche agrégée (Kopliku A., 2009). Elles peuvent être fusionnées et ordonnées de plusieurs manières :

- La phase de sélection traite la recherche d'information qui est potentiellement utile. Les K premiers résultats d'un moteur de recherche est un choix parmi d'autre.
- La phase de filtrage permet de supprimer l'information inutile. Par exemple, elle peut traiter la redondance de l'information. La phase de sélection et de filtrage correspondent à l'intuition que : « la recherche agrégée présente la majeure partie des informations utiles et le moins d'information inutile.

- Et puisque le contenu ne peut pas être aléatoirement placé dans le document agrégé, une phase d'organisation est nécessaire pour visualiser le contenu d'une façon approprié. Certains contenus doivent être représentés dans une liste, d'autre doit être groupée et ainsi de suite. L'organisation et la relation entre contenu recherché est une partie cruciale dans la recherche agrégée.

3.2.4. Types de contenu

Il est important de distinguer différents types de contenu parce que la récupération et le processus d'agrégation dépendent fortement des types de contenu. Il y a beaucoup de choix. Quelques types en incluent d'autres. Ci-dessous une brève classification par type:

! Par type de média : texte

! Granules d'information : livre, article, chapitre, titre, page Web, mot

! Utilisation contextuelle : blog, revue,

! Champ : email, numéro de téléphone, adresse

! Par tag : dans les documents semi-structurés.

3.3. L'Agrégation dans la recherche d'information traditionnelle (RI)

La recherche agrégée commence à partir de la recherche du contenu et elle finit par le filtrage, la sélection et l'organisation. Beaucoup de travaux existent du côté de la recherche traditionnelle de contenu, mais l'agrégation n'a pas été en grande partie étudiée. Peu de travaux existent dans la littérature et la formalisation est presque absente. Pendant la conférence d'ACM SIGIR 08 (Murdock et al., 2008), un atelier a été tenu particulièrement pour la recherche agrégée. Sushmita et al. (Sushmita et al., 2008)proposent de visualiser comme résultats de recherche, des pages de synthèses élaborées en groupant les documents renvoyés par un moteur de recherche.

Les solutions commerciales comportent déjà les fonctionnalités de recherche agrégées. Nous pouvons les trouver dans des contextes spécifiques tels que la recherche de produit ou la recherche d'endroit. Par exemple, Wize.com offre la recherche de produit avec des résultats qui sont obtenus comme agrégation de plusieurs sources (Michael, 2008). La recherche d'endroits de Google ajoute des numéros de téléphone, et des pages Web si disponibles en plus du résultat de carte.

L'agrégation semble être la tendance dans la recherche sur Web aussi, où la recherche par type de contenu tel que des cartes, des actualités, etc, est ajouté à la liste de pages Web présentées dans la page principale de résultats.

Il existe aujourd'hui deux approches principales. Dans la première, le contenu apparaît sous forme de listes HTML et le tout est ordonné selon un algorithme d'appariement. La recherche de Google reflète cette approche. L'autre approche implique une nouvelle disposition de résultats avec des espaces normalisés pour chaque type de contenu.

3.3.1 Recherche de Google

Google, a présenté une approche différente dans la recherche sur Web depuis 2007. Il applique un appariement sur les différents types (actualités, pages Web) et les résultats de différents moteurs de recherche verticaux de Google sont rangés dans la liste principale de résultat (voir la fig. 3.1). Le critère d'appariement reste la pertinence. Ceci signifie que du contenu de type différent est comparable par rapport à la pertinence vis-à-vis de la requête. Une approche précédente à celle actuellement utilisée était d'ajouter des résultats verticaux de recherche au-dessus ou au-dessous des résultats principaux une fois évalué comme pertinents.

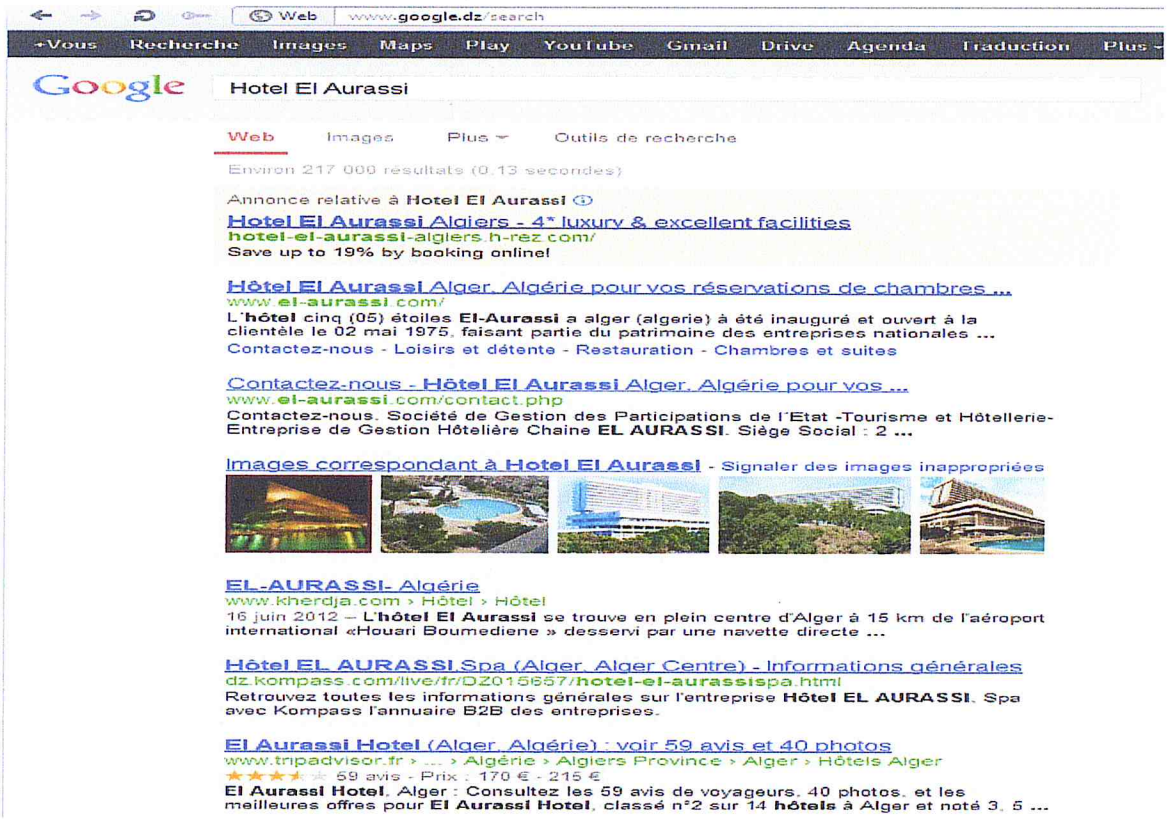


Fig.3.1 : *Résultats de recherche sur « Google » pour la requête « Hôtel El Aurassi ».*

3.3.2 Google News

Google News, est un bon exemple pour l'agrégation de l'information. Il agrège des actualités venant de différentes sources dans des histoires. Des actualités sont cherchées d'une liste d'environ 4500 emplacements pour l'anglais.

Il n'est pas difficile de trouver l'agrégation ici. Tout d'abord, les résultats ne sont pas une liste d'actualités simples, mais des groupes ou clusters (voir la fig. 3.2). Chaque cluster essaye de représenter une seule histoire. Le cluster est intitulé par un des articles les plus représentatifs. Il y a une récapitulation courte de cet article avec deux ou trois phrases et il y a ci-dessous une liste d'autres articles de la même histoire. En fin de compte, il y a la liste de tous les articles appropriés dans le cluster. Google News applique l'élimination de la redondance : tandis que les reproductions proches d'un article ne sont pas montrées si détectées, il y a une option pour l'utilisateur dans le cas où il les veut. Le moteur doit ranger les clusters et le contenu dans le même cluster.

La pertinence n'est pas le seul critère d'ordonnancement et de groupement. Le temps est également une dimension fondamentale. Les actualités récentes et fraîches sont préférées aux vieilles. Mais, l'utilisateur peut passer en revue par la chronologie. Il peut choisir la période où il est intéressé. Nous pouvons considérer que des actualités dans ce cas-ci sont groupées par des intervalles.

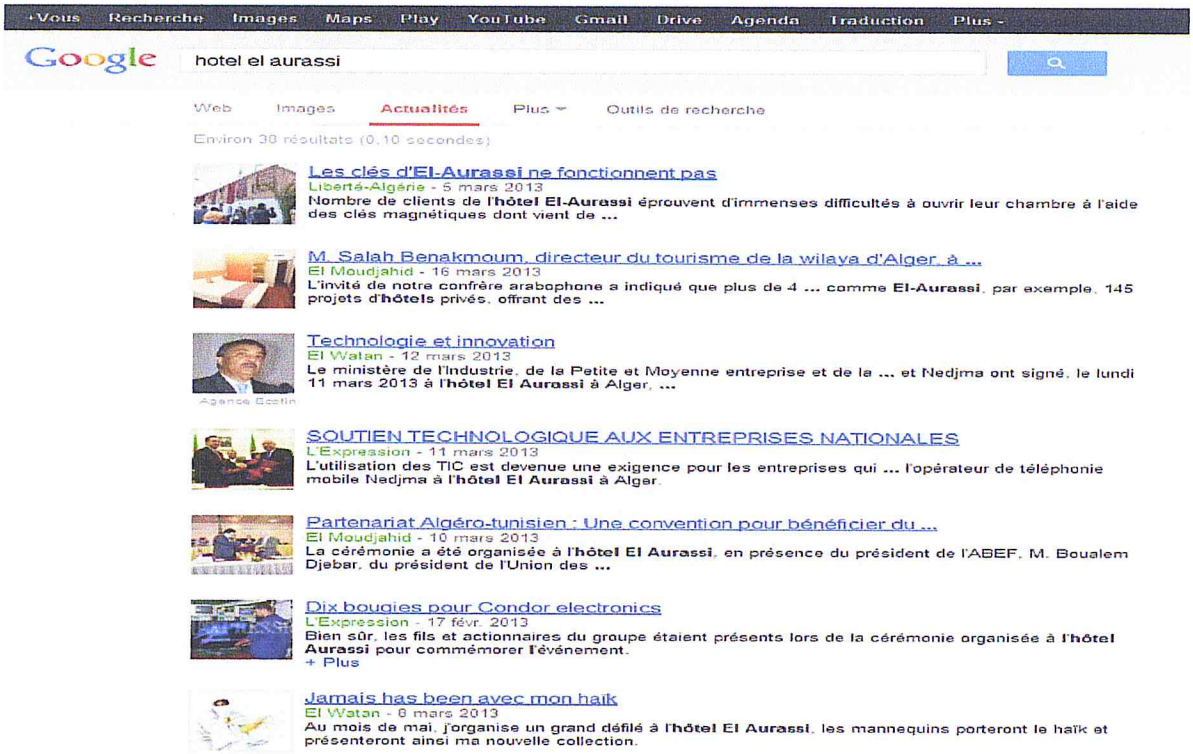


Fig. 3.2: *Résultats de recherche sur « Google News » pour la requête « Hôtel El Aurassi ».*

3.4. L'Agrégation dans la recherche d'information structurée (RIS)

Concernant les SRI dans des documents XML, ils sont capables de trouver des éléments de différents types et de différentes granularités (Torjmen et al., 2009). En fait, dans le contexte de la recherche XML, il a été montré qu'en retournant plusieurs éléments ensemble permet de donner plus de satisfaction aux utilisateurs que de renvoyer des éléments séparés. Une étude de cas intéressante vient d'INEX (Fuhr et al., 2008).

Dans l'agrégation des documents structurés, il s'agit d'affronter des problématiques liées à la pertinence des éléments XML, la diversité du contenu retourné, la couverture des différents aspects (sujets) de la requête formulée, la non-redondance (les éléments retournés ne véhiculent pas la même information) ainsi qu'à leur granularité. L'agrégation de ces éléments représente un document composite (agrégat). Cet agrégat cherche à produire un contenu agrégé reprenant les informations les plus pertinentes et non-redondantes qu'un utilisateur pourra trouver dans un corpus dedocuments XML en rapport avec sa requête.

On trouvera des SRI qui commencent à présenter les résultats d'une requête sur des documents XML sous forme de résumés. Par exemple, eXtract **(Huang et al., 2008)** est un SRI qui génère des résultats sous forme des fragments XML à partir de différents documents.

Le problème d'agrégation des éléments XML n'est pas encore traité et détaillé dans la littérature sauf le travail de **(Naffakhi.N et al., 2009)** et de **(Bessai, 2005)**. Ces deux travaux se basent sur le même principe : un modèle pour la RIS basé sur les réseaux

Bayésiens (RB). Les relations de dépendances entre requête, termes et éléments sont

quantifiées par des mesures de probabilité dans **(Naffakhi.N et al., 2009)** et des mesures de possibilités dans le modèle de **(Bessai, 2005)**. Dans ce modèle, la requête de l'utilisateur déclenche un processus de propagation pour trouver des éléments pertinents.
Ainsi, au lieu de récupérer une liste d'éléments qui sont susceptibles de répondre à la requête, l'objectif est d'agréger dans un seul document composite des éléments pertinents. La structure réseau Bayésien fournit une manière naturelle de représenter les liens entre les éléments du corpus de documents XML et leurs contenus. Il est à noter que ces deux modèles n'ont pas été évalués sur des collections de documents. Les résultats sont illustrés à travers des exemples.

3.4.1. Modèle possibiliste pour la recherche d'information XML

Le modèle que propose (Bessai et al., 2005) est représenté par un réseau possibiliste d'architecture illustrée par la figure 3.3. D'un point de vue qualitatif le graphe permet de représenter les nœuds documents, termes d'indexation, nœuds (balises d'un document XML). Les liens entre les nœuds permettent de représenter les relations de dépendances entre les différents nœuds.

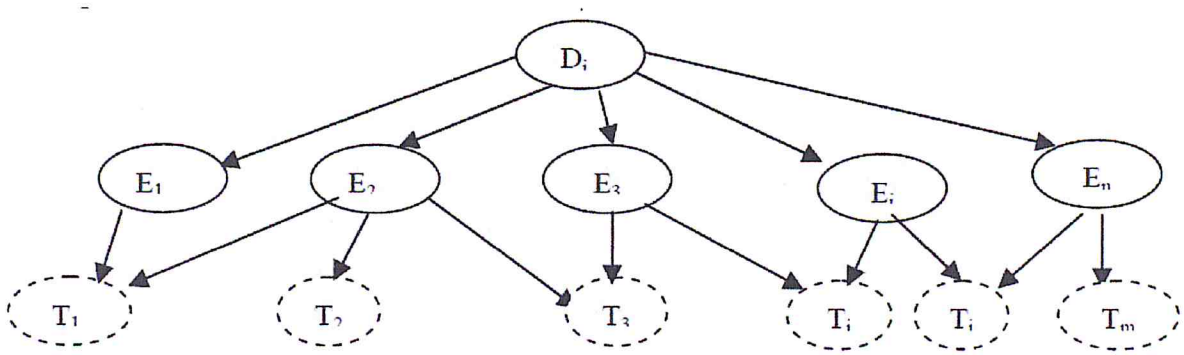


Fig.3.3 : *Architecture générale du modèle proposé*

3.4.1.1 Description du modèle

Les nœuds documents représentent les documents de la collection. Chaque nœud document Di, représente une variable aléatoire binaire prenant des valeurs dans l'ensemble dom(Di) = {di, ¬di}, où di représente 'le document Di est pertinent pour une requête donnée', et ¬di représente 'le document Di n'est pas pertinent pour la requête donnée'.
Les nœuds E1, E2, …En, représentent les balises du document Di. Chaque nœud Ei, représente une variable aléatoire binaire prenant des valeurs dans l'ensemble dom(Ei) = {ei, ¬ei}. L'instanciation Ei = ei signifie que l'élément 'ei' est pertinent pour la requête; Ei = ¬ei signifie que l'élément 'ei' est non pertinent pour la requête.
Les nœuds T1, T2, …, Tm sont les nœuds termes. Chaque nœud terme Ti représente une variable aléatoire binaire prenant des valeurs dans l'ensemble dom(Ti) = {ti, ¬ti,} où L'instanciation Ti = ti signifie que le terme 'Ti' est représentatif du nœud père auquel il est rattaché, Ti = ¬ti signifie que le terme 'Ti' est non représentatif du nœud père auquel il est relié. Il faut noter qu'un terme est relié aussi bien au nœud qui le comporte ainsi qu'à tous les ascendants de ce dernier.

Le passage du document vers la représentation sous forme de réseau possibiliste se fait de manière assez simple. Il consiste à ramener tous les nœuds (balises du document) au niveau des variables Ei. Les valeurs qui seront assignées aux arcs de dépendances entre nœuds termes - nœuds balises et nœuds balises - nœuds document dépendent du sens que l'on donne à ces liens.
Chaque variable structurelle Ei, Ei∈E = {E1, E2…, Em}, dépend directement de son nœuds parent qui est le nœud racine D, dans le réseau possibiliste du document.
Chaque variable de contenu Ti, Ti ∈T = {Ti, …, Tk} dépend uniquement de sa variable structurelle (élément structurel ou balise). Il faut également noter que la représentation fait apparaître un seul document. En fait on considère que les documents sont indépendants les uns des autres donc on peut raisonner en considérant le sous réseau qui représente le document que l'on traite.
Nous notons par T(E) (resp. T (Q)) l'ensemble des termes d'indexation des éléments du document (resp. de la requête).
Les arcs sont orientés et on en distingue deux types :
– Les liens termes-balises : Ces liens relient chaque nœud terme $Ti \in T(E)$ à chaque nœudEi ou il apparaît.
– Les liens balises-document : Ces liens relient chaque nœud balise de E au document qui le comporte, en l'occurrence D dans notre cas. Nous discuterons dans la section 4.3, l'interprétation que nous donnons à ces différents liens et la manière de les quantifier.

3.4.1.2 Evaluation d'une requête

L'évaluation d'une requête est effectuée par à un processus de propagation (Benferhat, 1999) (Borgelt, 2000). Elle consiste à injecter une nouvelle évidence à travers les arcs activés du réseau pour rechercher les documents et les éléments pertinents par rapport à la requête. Comme nous l'avons souligné précédemment, nous modélisons la pertinence selon deux dimensions : la nécessité et la possibilité de pertinence.

3.5. Conclusion

Dans ce chapitre, nous avons présenté la recherche d'informations agrégés comme nouveau secteur de recherche qui nécessite une formalisation théorique. Et nous avons vu que, quelques fonctionnalités du principe d'agrégation ont été déjà utilisées dans la recherche traditionnelle sur web. Mais concernant le domaine des documents structurés (XML), les travaux sont très peu, et même les procédures et les mesures d'évaluation n'existent pas pour le moment.
Par la suite nous proposons une approche qui se situe à la jonction de la recherche des éléments pertinents à partir de documents XML et leur agrégation dans un même résultat. Mais la différence c'est qu'on propose une agrégation des résultats XML par document, c'est-à-dire qu'on revoie à l'utilisateur une liste ordonnée d'éléments agrégés, où l'agrégat retourné contiendra les informations pertinentes du document associés.

CHAPITRE 4

Proposition pour l'agrégation des résultats dans la recherche d'information structurée.

Chapitre 4 :
Proposition pour l'agrégation des résultats dans la recherche d'information structurée.

4.1 Introduction

L'information est principalement disponible sous forme de texte. Obtenir des réponses pertinentes, organisées, triées et exploitables, est un besoin quotidien. L'information textuelle ne peut pas être gérée comme des données. En effet, il ne s'agit pas d'accéder à une information structurée, identifiée par des chemins d'accès prédéfinis comme dans les bases de données. Il faut aller beaucoup plus loin : l'information recherchée doit pouvoir être identifiée par les sujets qu'elle recouvre, c'est-à-dire son contenu textuel.
Différents outils et méthodes ont été appliqués en vue d'obtenir d'autres interfaces de consultation, prenant en compte de manière efficace la sémantique des requêtes et des documents, en particulier les méthodes de classification sous diverses formes (taxonomie ou méthodes neuronales).
L'objectif du présent travail est de réaliser un système de recherche documentaire basé sur des algorithmes de recherche puissants, exacts et efficaces.qui permettent de constituer un système adapté à la classification et la consultation de documents. En utilisant Apache Lucene, un moteur de recherche textuelle.

Nous exposons dans cette partie du mémoire, notre travail de développement d'une application, ayant la capacité pour l'agrégation des résultats.
Notre objectif est de renvoyer un e agrégat. Il est composé d'éléments non redondants, issus d'un même document et censé représenter (Comporter) tous les éléments permettant de mieux répondre à la requête.
Cet agrégat est Dans notre cas, Construits à partir de la liste comportant les éléments élémentaires Renvoyés en réponse à une requête.

4.2. Qu'est-ce que c'est Apache Lucene ?

Il s'agit d'un projet de la fondation Apache (Sekiguti ,2006) qui fournit un moteur de recherche rapide, basée sur l'indexation clé-valeur. Développé depuis 2000, le projet a atteint sa maturité en plein sens du terme. Il est composé de plusieurs concepts :
- documents : endroits qui contiennent les termes recherchés.
- score : signifie pertinence des résultats.
- champ : (field) correspond à une portion de données qui compose chaque document. Le champ peut ensuite être pris en compte lors de la recherche.
- terme : (term) similaire au champ, il définit une chaîne de caractères recherchée dans les documents pendant la recherche.

4.2.1 Utilisation et atouts de Lucene

- Il propose de nombreux types de requêtes, tels que PhraseQuery, WildcardQuery, RangeQuery, FuzzyQuery, BooleanQuery, et d'autres encore.
- Il permet l'analyse d'expressions riches du type de celles qui sont saisies par des êtres humains
- Il utilise des algorithmes de recherche puissants, exacts et efficaces.
- Il permet aux utilisateurs d'utiliser des algorithmes d'appariement et d'étendre le comportement de la recherche en utilisant des tri personnalisés, des filtres et l'analyse d'expressions.
- Il utilise un mécanisme de verrouillage basé sur les fichiers pour empêcher les modifications d'index concurrentes.
- Il permet d'indexer et de rechercher simultanément.

4.2.2 Construction d'application avec Lucene

Comme le montre la Figure 1, construire une application de recherche complète avec Lucene (Hatcher et al., 2004) implique principalement l'indexation et la recherche des données, et l'affichage des résultats d'une recherche.

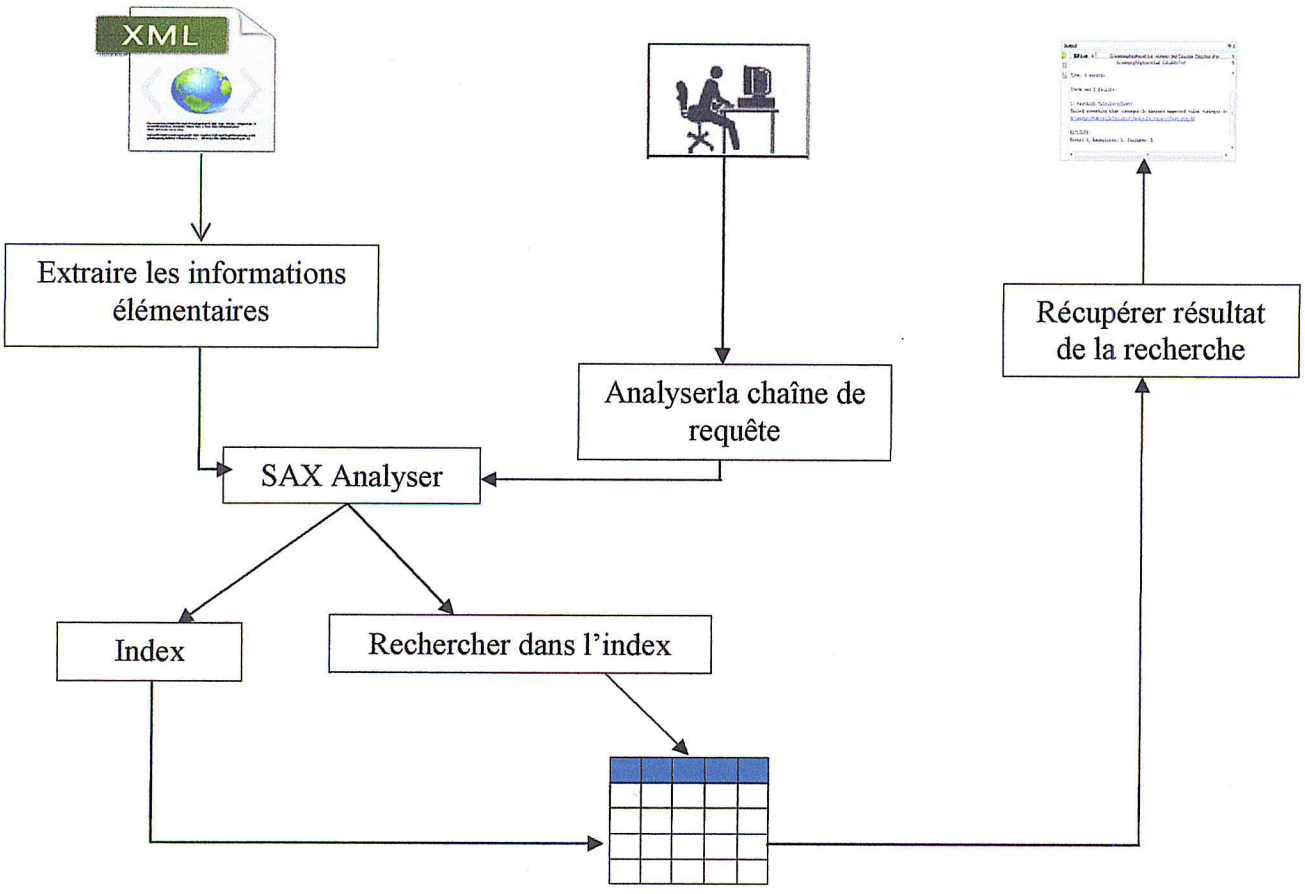


Fig.4.1 : *Les étapes de la construction d'applications avec Lucene*

4.2.3 Indexer les documents XML

XML permet de définir la structure du document uniquement, ce qui permet d'une part de pouvoir définir séparément la présentation de ce document, d'autre part d'être capable de récupérer les données présentes dans le document pour les utiliser.

L'indexation est une des étapes les plus importantes dans notre projet parce qu'elle crée l'un des plus primordiales des composants de notre projet qui repose sur les informations collectées par les documents ajouté par l'Administrateur. Toutefois la récupération des données encapsulées dans le document nécessite un outil appelé *analyseur syntaxique* (en anglais *parser*), permettant de parcourir le document et d'en extraire les informations qu'il contient, ainsi pour indexer les documents ajoutés on passe par plusieurs étapes :

4.2.3.1Extraction des unités fondamentales de recherche

En utilisant **SAX**(*Simple API for XML)* est la principale interface utilisant l'aspect événementiel), est une API basées sur un mode **événementiel, SAX** permet L'***analyse*** est la conversion des données textuelles en unités fondamentales de recherche, appelées ***termes***. Pendant l'analyse, le texte subit plusieurs opérations: extraction des mots, ***lemmatisation*** et la ***pondération*** des mots les plus courants et de la ponctuation, passage en minuscules, etc. L'analyse du texte est effectuée avant l'indexation et l'analyse des requêtes. Elle convertit les données textuelles en tokens(une *entité lexicale)* et ces tokens sont ajoutés en tant que termes à l'index Lucene.

L'API SAX :

- Définit des triggers qui se déclenchent sur certaine balises.
- Adaptée aux applications qui extraient de l'information d'un document XML.

Ainsi, une application basée sur SAX peut gérer uniquement les éléments dont elle a besoin sans avoir à construire en mémoire une structure contenant l'intégralité du document.

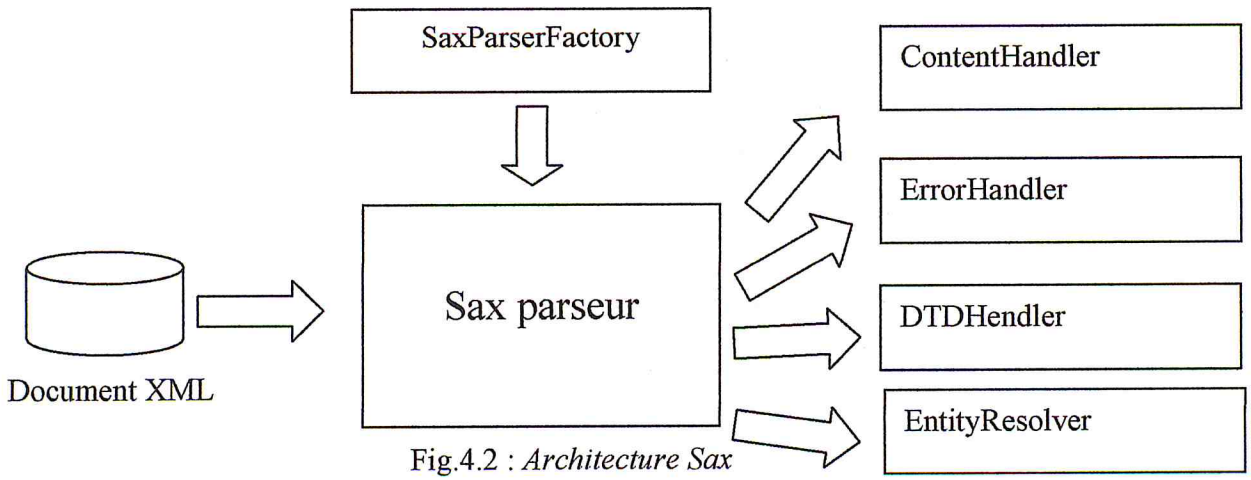


Fig.4.2 : *Architecture Sax*

L'API SAX définit les quatre interfaces suivantes :

- **DocumentHandler** possédant des méthodes renvoyant des événements relatifs au document :
  - *startDocument()* renvoyant un événement lié à l'ouverture du document
  - *startElement()* renvoyant un événement lié à la rencontre d'un nouvel élément
  - *characters()* renvoyant les caractères rencontrés
  - *endElement()* renvoyant un événement lié à la fin d'un élément
  - *endDocument()* renvoyant un événement lié à la fermeture du document
- **ErrorHandlerp** possédant des méthodes renvoyant des événements relatifs aux erreurs ou aux avertissements
- **DTDHandler** renvoie des événements relatifs à la lecture de la DTD du document XML
- **EntityResolver** permet de renvoyer une URL lorsqu'une URL est rencontrée

4.2.3.2 Processus d'indexation

L'*indexation* est le processus de conversion des données textuelles vers un format qui facilite une recherche rapide. Une analogie simple est celle de l'index que l'on trouve à la fin d'un livre: cet index indique la localisation des différents sujets qui sont traités dans le livre.

On stocke les données en entrée dans un ***tableau*** de données dynamique appelée index *inversé*, qui est lui même stocké dans le système de fichiers ou en mémoire comme un ensemble de fichiers d'index. La plupart des moteurs de recherche Internet utilisent un index inversé. Cela permet aux utilisateurs d'exécuter des recherches par mot-clé rapides et de trouver les éléments pertinents répondant à une requête données. Avant que la donnée textuelle ne soit ajoutée à l'index, elle est traitée par un analyseur.

4.2.3.3 Rechercher des données indexées

Rechercher c'est examiner l'index pour y trouver des mots puis obtenir éléments. Il est facile d'ajouter des possibilités de recherche à une application en utilisation l'API de recherche de Lucene.

Query est la classe abstraite pour les requêtes. Rechercher un mot ou une expression se fait en l'encapsulant dans un ou plusieurs termes, en les ajoutant à un objet requête et en passant cette requête à la méthode de recherche de IndexSearcher.

Lucene fournit plusieurs implémentations de requêtes, tels que TermQuery, BooleanQuery, PhraseQuery, PrefixQuery, RangeQuery, MultiTermQuery, FilteredQuery, SpanQuery, etc. La section suivante discute des principales classes de requête de l'API de requêtage de Lucen(Hardt, 2004).

4.3. Agrégation des éléments pertinents pour la recherche XML

Nous nous intéressons à la recherche agrégée dans des documents structurés XML. Pour cela, nous proposons un modèle de recherche d'information. Dans ce modèle, la requête de l'utilisateur

déclenche un processus de propagation pour trouver des éléments. Ainsi, au lieu de récupérer une liste d'éléments potentiellement (ou partiellement) pertinents vis-à-vis de la requête, notre objectif est de rassembler dans un agrégat des éléments pertinents, non-redondants et complémentaires susceptibles de mieux répondre à la requête.

On parcourt la liste ***d'éléments pertinents*** et on regroupe les nœuds appartenant au même document. On forme ainsi ce que l'on appelle les ***agrégats bruts***.

L'exemple suivant donne un aperçu sur la façon d'obtenir les agrégats bruts à partir des éléments pertinents, on suppose que le parcours de la liste *lr* a permet d'identifier que les nœuds (n_1, n_{25}, n_{101} et n_{999}) appartiennent au même document *d5*, et les nœuds (*n2, n3* etn_{99}) appartiennent au même document d_{11}, etc… on aura alors les deux agrégats bruts suivants

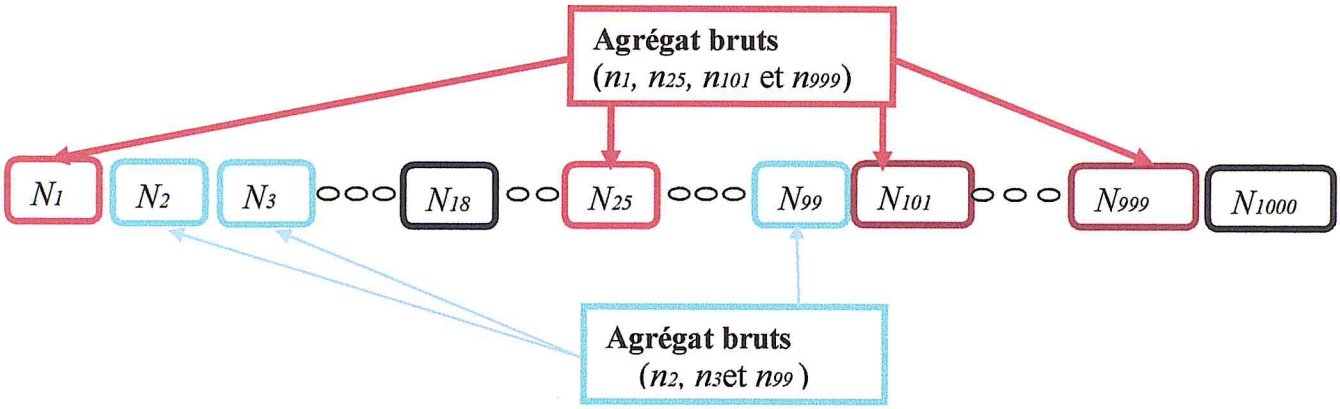


Fig.4.2: Création des agrégats bruts à partir de la liste initiale lr.

4.3.1Elimination des nœuds

Après la création de tous les agrégats bruts on procède maintenant à une épuration de ces agrégats. Ceci revient à essayer d'éliminer les nœuds susceptibles d'affaiblir le score de l'agrégat.

Les SRI structurées actuels tentent de trouver l'élément (le sous arbre) le plus petit possible répondant à la requête. Dans notre cas, nous traduisons cette contrainte au niveau de l'agrégat. Nous proposons donc d'éliminer les nœuds affaiblissant le score de l'agrégat, en d'autre terme, nous proposons l'agrégat contenant le plus petit nombre d'éléments possible répondants le mieux à la requête.

Pour cela on définit la fréquence d'apparition de l'élémentF_p, qui est utilisée pour contrôler l'écart entre la fréquence d'apparition$F_p$du meilleur élément dans l'agrégatA_k, *noté* $F_{pmax\ ak}$et la fréquence d'apparition du faible élément dans le même agrégat soit $F_{pmin\ ak}$

Tel que :

- $0 < F_p < 1$

- Si F_p est élevé, alors l'agrégat va contenir plus d'éléments parce que d'autres éléments de l'agrégat brut du document avec une fréquence d'apparition $F_p$pas importante vont être rajoutés à l'agrégat.
- Si on diminueF_p, alors l'agrégat va contenir moins d'éléments parce que des éléments de l'agrégat brut du document avec une fréquence d'apparition $F_p$enlevée de l'agrégat.

Nous proposons alors d'éliminer des nœuds (éléments) si leur fréquence d'apparition est inférieur a la fréquence d'apparition de l'élément optimal. On élimine ainsi les éléments ayant peu d'intérêts dans l'agrégat. Plus précisément, pour chaque élément n_i de l'agrégat brut A_k une fréquence d'apparition F_p comme suit :

	Document			
Terme	D1	D2	D3	D4
Note	13	0	20	8
Gamme	07	09	11	06
Octave	0	15	20	16
Accord	17	19	22	0

Tab.3.1 Nombre de fois que le terme est employé dans un document nbr t.

2436	1652	1896	2012

Tab.3.2 Nombre de termes total dans chaque document $nbr\ t_{max}$.

- Formule de calcule de la fréquence d'apparition d'un terme :

$$F_P - \frac{nbr\ t}{nbr\ t_{max}}$$

	Document			
Terme	D1	D2	D3	D4
Note	0.0054	0	0.010	0.004
Gamme	0.0029	0.0054	0.0058	0.003
Octave	0	0.0091	0.0105	0.008
Accord	0.007	0.012	0.011	0

Tab.3.3 Fréquence du terme dans le documentF_p.

4.4 Spécification semi-formelle des besoins

Une étude approfondie des besoins fonctionnels s'avère indispensable avant d'entamer la conception, afin d'obtenir de manière plus formelle une vue globale sur les exigences de l'application. Cette partie présente alors une modélisation de ces besoins en faisant recours aux concepts fondamentaux du langage de référence UML (Unified Modeling Language). À savoir le diagramme de cas d'utilisation.

UML ne sert ici qu'à formaliser les besoins, c'est-à-dire à les représenter sous une forme graphique suffisamment simple pour être compréhensible par toutes les personnes impliquées dans le projet. N'oublions pas que bien souvent, le maître d'ouvrage et les utilisateurs ne sont pas des informaticiens. Il leur faut donc un moyen simple d'exprimer leurs besoins. C'est précisément le rôle d'UML. Ils permettent de recenser les grandes fonctionnalités d'un système (Barbier, 2005).
Après avoir donné un petit aperçu sur l'utilité, et ce que peut représenter un diagramme de cas d'utilisation, on vous expose dans la figure 4.2 le diagramme de cas d'utilisation de notre futur application, les principaux cas d'utilisations seront exprimés plus en détails par la suite.
On a utilisé le logiciel Edraw pour modéliser les différents diagrammes

4.4.1 Digrammes de cas d'utilisation

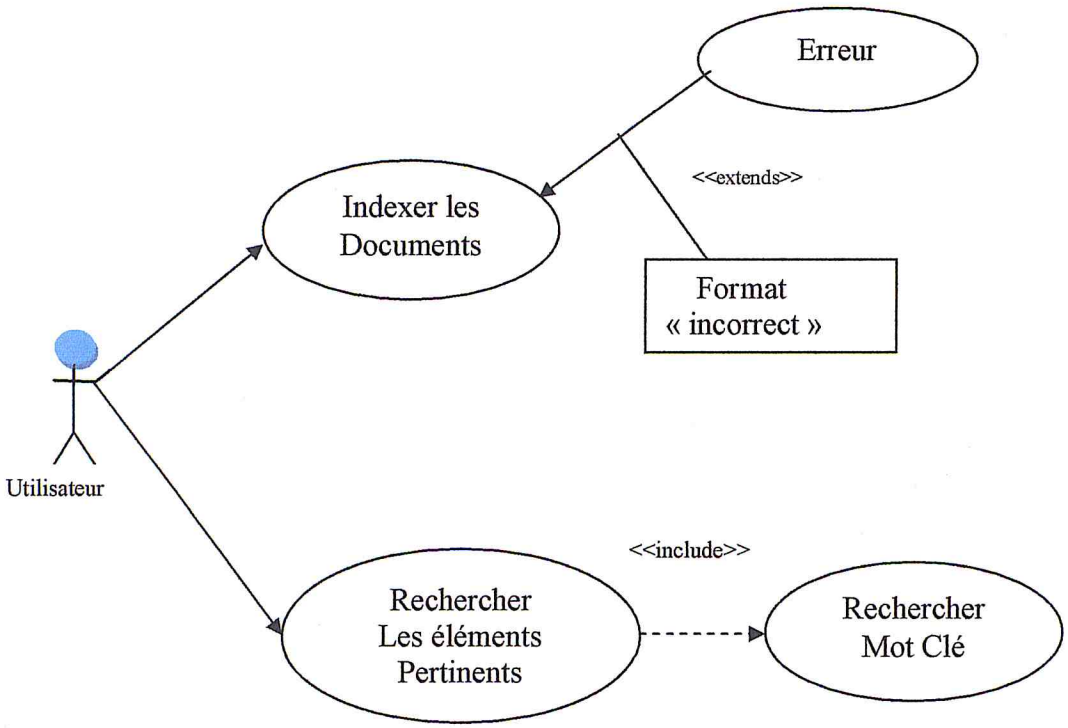


Fig.4.3 : Digrammes de cas d'utilisation

Cas d'utilisation	Description
Recherche	L'utilisateur accède à la page de la recherche, tape la requête et valide puis il reçoit le résultat (l'ensemble des éléments (Agrégat)) dans un temps minimum d'après sa requête.
Ajouter des documents XML	L'utilisateur accède à la page d'ajout et choisi le document à ajouter depuis son disque dure et valide son choix et le document s'il est de format XML sera envoyé.

4.4.2 Digrammes de Séquence

Nous décrivons dans la figure 4.3 par l'intermédiaire d'un diagramme de séquence UML, le processus d'ajout d'un Document XML. Chaque flèche continue dans le diagramme représente un appel de fonction (une flèche discontinue présente son rappel) ces fonction sont enclenché à l'intérieur des objets Interface, Système, Index ainsi qu'un objet modélisant un utilisateur.

4.4.2.2Description du processus de d'indexation

Nous décrivons dans la figure 4.4 par l'intermédiaire d'un diagramme de séquence UML, le processus d'indexation.

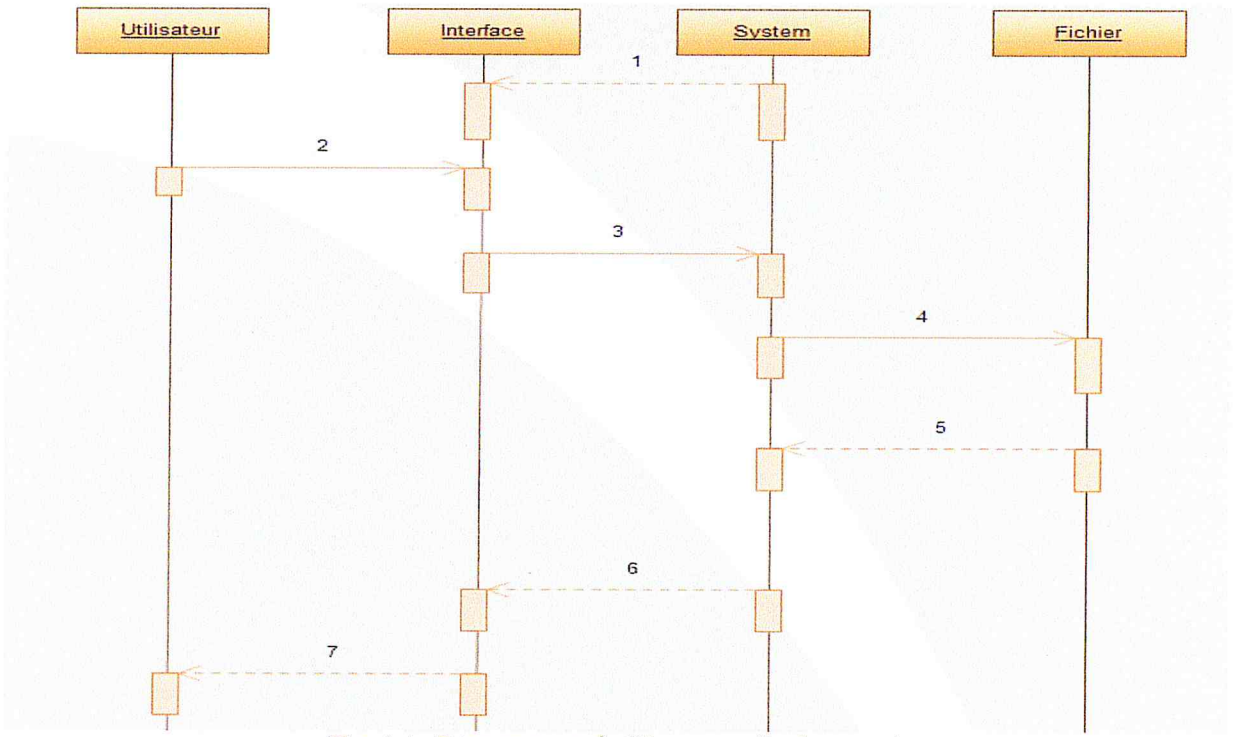


Fig.4.4 :Digrammes de Séquence (Indexation)

1. Le système affiche l'interface.
2. L'utilisateur clique sur l'évènement Indexing.
3. La requête est envoyée au système.
4. Le système crée et envoie l'index dans un fichier.
5. Il renvoie la réponse au système.
6. Renvoyer le résultat à l'interface (document indexer).
7. Les Document sont indexés.

4.4.2.2Description du processus de Recherche

Nous décrivons dans la figure 4.5 par l'intermédiaire d'un diagramme de séquence UML, le processus de Recherche.

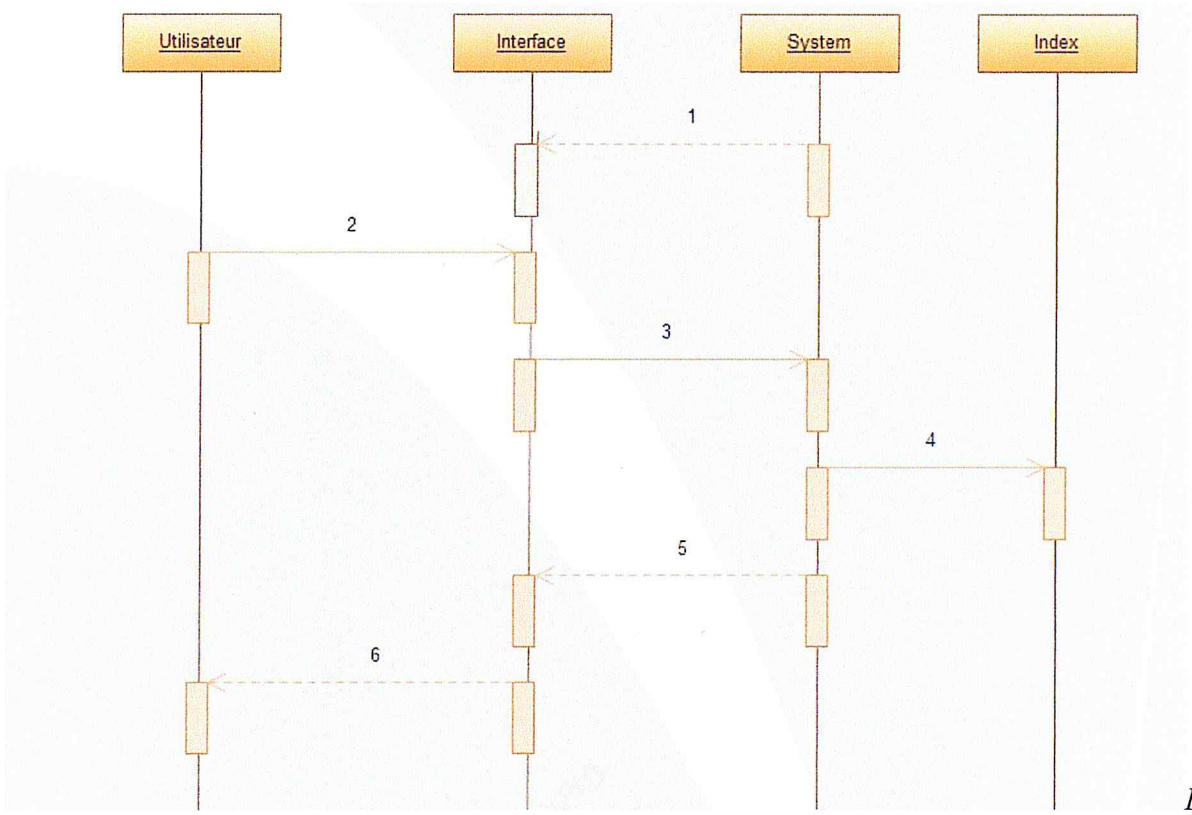


Fig.4.5 :Digrammes de Séquence (Recherche)

1. Le système affiche la page de la recherche.
2. L'utilisateur saisi la requête et valide.
3. La requête est envoyée au système.

4. Rechercher dans l'index.
5. Renvoyer le résultat à l'interface (les agrégats).
6. Afficher le résultat.

4.4.3 Digrammes de Classe

Nous décrivons dans la figure 4.6 par l'intermédiaire d'un diagramme de Classe UML, le processus de fonctionnement de notre projet.

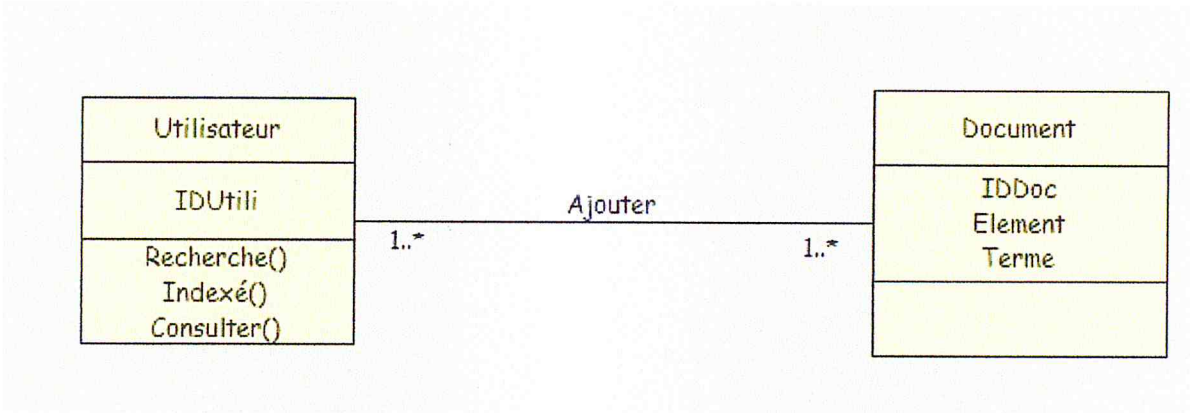


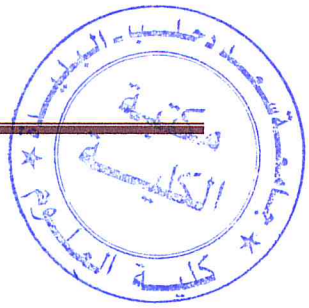
Fig.4.6:Digrammes de Classe

4.5 Conclusion

Nous avons présenté dans ce chapitre une proposition pour l'agrégation des résultats dans le contexte de la recherche d'information XML.
L'outil de développement Lucene a été utilisé pour concevoir un système de classification automatique et de consultation adapté aux données textuelles.
Les recherches s'effectuent grâce à une classe abstraite qui possède différentes méthodes de recherche, SercheSampler est une sous-classe qu'on utilise pour la recherche dans les index stockés dans un répertoire donné. La méthode retourne une collection d'éléments classés selon leurs scores calculés. On calcule la fréquence d'apparition pour chaque élément correspondant à une requête ainsi l'agrégat contiendra les meilleurs éléments du document, supposés répondre à la requête.

CHAPITRE 5

Implémentation du système



Chapitre 5 Implémentation du système

5.1 Introduction

A l'époque où les millisecondes comptent pour tous, l'optimisation des moteurs de recherche est devenue un enjeu majeur. L'un des projets de la fondation Apache, Lucene, répond parfaitement aux attentes des développeurs. Et son serveur écrit en Java, est synonyme d'un vrai gain des performances aussi bien qu'au niveau de la mise en place qu'au niveau des résultats.

Nous aborderons dans ce chapitre les phases restantes de notre cycle de développement, en représentant les différentes étapes que nous avons réalisé pour la mise en œuvre de notre application qui a pour rôle la recherche d'information agrégée des documents XML.

5.2 Codage

Arrivé à ce stade, il ne reste qu'à commencer à écrire notre code en se basant sur les résultats obtenus des chapitres précédents.

5.2.1 Langage et outil de développement

Nous avons choisie comme développement pour notre application un langage orienté objet, qui a été jugé en conformité avec la conception que nous avons établis. il s'agit du langage JAVA, pourvu d'une grande sécurité, la richesse de ses bibliothèques, son adaptation à plusieurs plateformes, la qualité présentée par ses composantes graphiques (Swing). Une grande partie de sa syntaxe est empruntée de C et C++.

Pour ce qui est du choix de l'outil de développement intégrés (IDE) pour java, notre choix à aboutit sur l'un d'entre eux considéré comme l'un des plus puissant IDE JAVA mis sur le marché "Eclipse".

Eclipse est un projet open source à l'origine développé par IBM pour ces futurs outils de développement. Le but est de fournir un outil modulaire capable non seulement de faire du développement en java mais aussi dans d'autres langages et d'autres activités. Cette polyvalence est liée au développement de modules réalisés par la communauté ou des entités commerciales.

5.2.2 Développement du projet

- Architecture du projet :

Notre projet est une application qui permet aux utilisateurs d'accéder à des documents XML ou chaque utilisateur effectue une recherche par des mots clés dans un moteur de recherche qui peut accéder à des informations stockées dans un index après un processus d'indexation des documents XML ajouté par l'utilisateur.

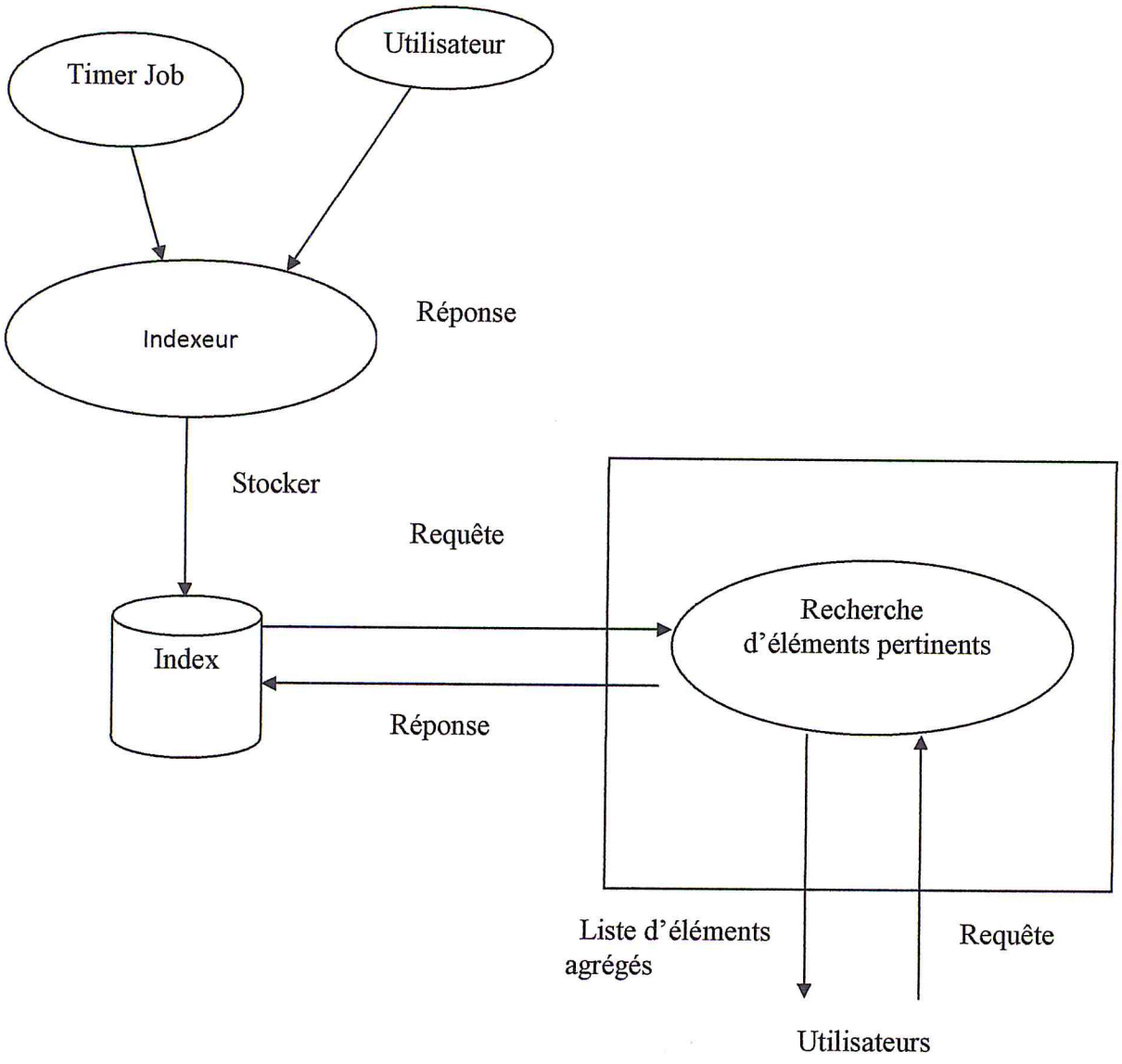


Fig.5.1:Architecture du projet

- APIs JAVA utilisées

Plusieurs APIs standard et non standard en été utilisés pour le développent de notre application.

-APIs Standard JAVA

Il nous a été indispensable pour développer notre application de prendre comme soutiens les APIs standard fournis par la machine virtuelle Java, ces derniers nous ont été utile pour le développement des interfaces d'interaction avec l'utilisateur, le tableau 3.1 décrit ces APIs.

API	Description
Swing	Inclut un grand nombre d'interface graphique, pouvant satisfaire la plus part des besoins de développent en cette matière, l'avantage de cette API est que ces composants graphique sont dite légères est n'occupent donc pas énormément de ressources (processeur, carte graphique) pour leurs bons fonctionnent,
Event	Sert comme capteur d'événements déclenché par les interfaces graphique, ces événement son en relation direct avec les actions de l'utilisateur (click souris ou clavier, navigation curseur etc.).
Eclipse Workbench	la dernière couche graphique permettant de manipuler des composants, tels que des vues, des éditeurs et des perspectives

Tab.5.1 Description des APIs standards utilisées dans Notre application.

-APIs non Standard JAVA

Plusieurs APIs non standard ont été également utilisée, chacune avait un rôle définit, nous avons pris en considération pour la sélection de ces derniers, l'organisme qui les a développé, ainsi que des sondages de satisfaction du publique de développeurs en cette matière.

API	Description
Plug-ins	Un plugin ou plug-in (aussi nommé module d'extension, greffon ou plugiciel) est un logiciel qui complète un logiciel hôte pour lui apporter de nouvelles fonctionnalités. La plupart du temps, ces programmes sont caractérisés de la façon suivante : • Ils ne peuvent fonctionner seuls car ils sont uniquement destinés à apporter une fonctionnalité à un ou plusieurs logiciels. • Ils sont mis au point par des personnes n'ayant pas nécessairement de relation avec les auteurs du logiciel principal.

Timer Job	Timer Job est un programme qui va lancer l'indexation des fichiers dans une heure précise. On implémente la class RemindTask qui s'étend de la class TimerTask qui peut exécuter une tache prévue pour un temps ou l'exécution répétée par une minuterie. Et ainsi on implémente la class Reminder ou on va préciser l'heure et la date du lancement du programme d'indexation.

Tab.5.2 Description de quelques APIs externes utilisées.

5.3 Fonctionnement de l'application

1. Comment Indexer les documents XML ?
2. Fonctionnement du Timer Job ?
3. Comment rechercher les éléments pertinents dans l'index ?
4. Comment construire l'Agrégat (ensemble d'éléments pertinents agrégés) ?

5.3.1 Indexation

Pour créer rapidement des index, Lucene utilise une approche légèrement différente de la plupart des outils de recherche traditionnels. Plutôt que de maintenir un seul index, il bâtit plusieurs segments d'index, qu'il fusionne périodiquement. Ainsi, pour chaque nouveau document indexé, Lucene crée un nouveau segment qui sera rapidement fusionné avec d'autres segments plus grands, de manière que les recherches se fassent rapidement.

Les classes nécessaires à la création d'un nouvel index, ou à l'extension d'un existant, se trouvent dans les paquetages *org.apache.lucene.index* et *org.apache.lucene.analysis.standard*. Le premier contient l'outil de création de l'index tandis que le second accueille les analyseurs de texte.

- Analyse des entrées avec Saxparser :

  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 parle alors de *parseur XML*), l'analyseur permet de créer une structure hiérarchique contenant les données contenues dans le document XML.

Spécifié dans le constructeur IndexWriter, prend en charge l'extraction de ces jetons de texte qui doivent être indexé et élimine le reste.

La classe « org.apache.lucene.analysis.Analyzer » est naturellement au coeur du processus d'analyse et en particulier son unique méthode, disposant de de la signature tokenStream.

```
Analyzer analyzer = new Analyzer ;
IndexWriter writer = new IndexWriter(directory, analyzer,
IndexWriter.MaxFieldLength.UNLIMITED);
```

-Les interfaces du package org.xml.sax :

ContentHandler	évènements liés au contenu
DTDHandler	évènements liés à la DTD
ErrorHandler	récupération des erreurs
Locator	récupération de la position
XMLReader	Analyseur

- Les classes du package org.xml.sax.helpers

org.xml.sax.helpers.DefaultHandler
implementsorg.xml.sax.ContentHandler, implements
org.xml.sax.DTDHandler, implements
org.xml.sax.EntityResolver,
implementsorg.xml.sax.ErrorHandler

- La première tâche que devra remplir un `Analyzer` consistera à découper (tokeniser) le flux reçu par le **Reader**. A cet effet, on utilisera généralement un descendant de la classe « org.apache.lucene.analysis.Tokenizer» chargé de découper le texte en `Token`s, sur les espaces ou les signes de ponctuation par exemple.
- Une fois ces `Token`s bruts obtenus, il sera souhaitable de les transformer et/ou de les supprimer du flux (cas des mots-vides). Cette opération est réalisée grâce à des descendants de la classe« org.apache.lucene.analysis.TokenFilter»qui recevront un `TokenStream` en entrée, et qui a leur tour, émettront des `Token`s, éventuellement à destination d'autres **TokenFilter**S.

On obtient donc un flux de `Token`s qui vont être transformés en objets « org.apache.lucene.index.Term »stockés dans les index Lucene. Un objet **Term** comporte 2 propriétés principales :

- une représentation textuelle, accessible via la méthode **text()**. Comparer avec **Token.termText()**.
- le champ pour lequel on définit le terme, accessible via la méthode**field()**. Dans SDX, il reprendra naturellement l'attribut **name**de l'élément **<sdx:field/>.**

- Pour un terme donné, l'index stocke donc un numéro d'ordre de document, le nombre d'occurrences du terme dans le document, ainsi que les différentes positions du terme dans ce document.

Notre index est un tableau dynamique connu sous le nom d'index inversé. « privatestaticArrayList<String>*fields*= new arrayList<String>(); », et chaque index inversé est constitué d'un ou plusieurs Segments, chaque segment est un index en lui-même, détiens un sous ensemble de tous les documents indexés qui est enregistré dans un répertoire (Directory).

Lucene on peut fusionner l'ensemble des segments en un seul index.Optimiser l'index pour faire une recherche très rapide,

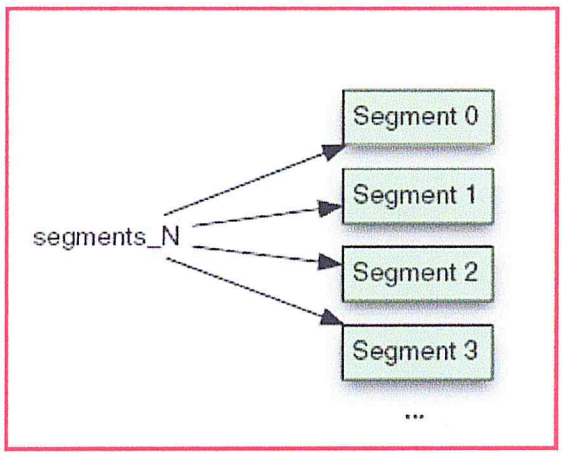


Fig.5.2:Architecture d'indexation

5.3.2 Timer Job

Timer Job est un programme qui va lancer l'indexation des fichiers dans une heure précise.
On implémente la class RemindTask qui s'étend de la class TimerTask qui peut exécuter une tache prévue pour un temps ou l'exécution répétée par une minuterie. Et ainsi on implémente la class Reminder ou on va préciser l'heure et la date du lancement du programme d'indexation.
On exécute en premier le main de la class Reminder qui fait appelle à son constructeur ou est définie le temps du lancement d'exécution du programme d'indexation, on utilise la class Calendar ou on affecte l'heure, la minute et même la seconde du lancement de notre programme, et on appelle la class Timer qui va grâce à sa méthode schedule(TimerTask task, Date time) préciser la tache à exécuter et le temps du lancement de cette tâche.

```
Calendar calendar = Calendar.getInstance ();
calendar.set (Calendar.HOUR_OF_DAY, 23);
calendar.set (Calendar.MINUTE, 51);
calendar.set (Calendar.SECOND, 0);
Date time = calendar.getTime ();

timer = newTimer ();
c
```

Dans notre cas la tache à exécuter dans l'heure précisée dans le timer est la class RemindTask ou on appelle la méthode run ( ) de TimerTask là ou on va définir le programme à exécuter.

```
publicvoid run () {

TikaConfigconfig = TikaConfig.getDefaultConfig ();
…
timer.cancel ();
}
```

5.3.3 ***Rechercher dans l'index :***

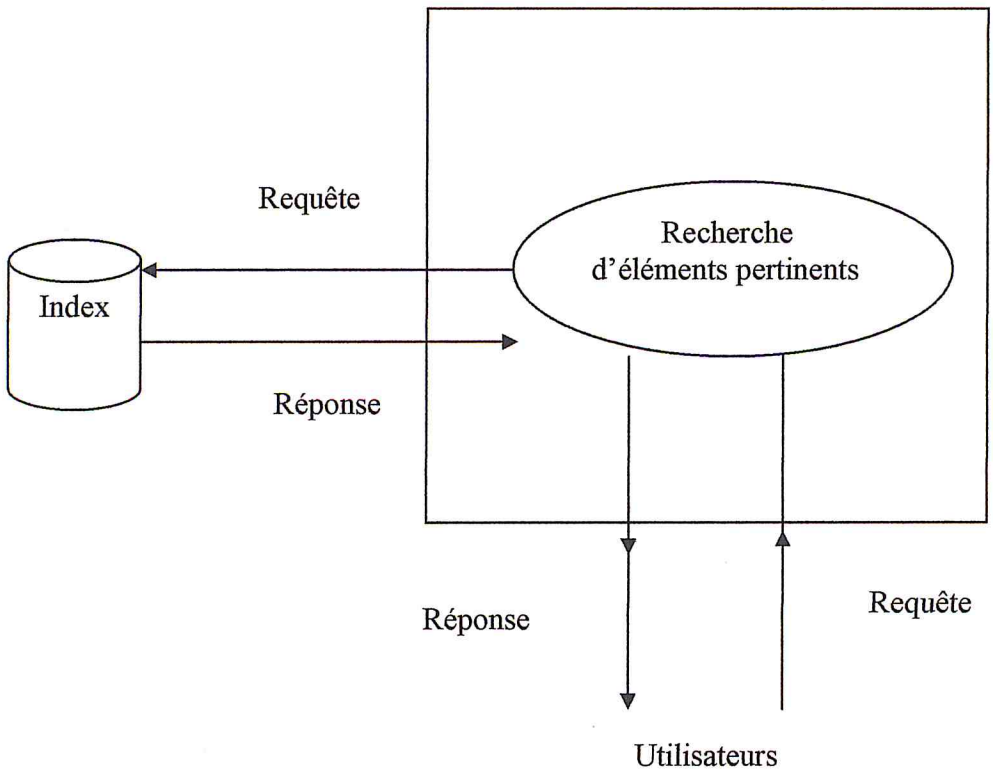


Fig.5.3: Comment rechercher dans l'index ?

5.4 Recherche d'élément agrégés

Les classes relatives aux recherches se situent dans les paquetages *org.apache.lucene.search* et *org.apache.lucene.queryParser*. Avant toute chose, le développeur doit charger un index sur lequel les requêtes seront appliquées.

- **Searcher**

Searcher est une classe abstraite qui possède différentes méthodes de recherche surchargées. IndexSearcher est une sous-classe couramment utilisée qui permet de rechercher dans les index stockés dans un répertoire donné. La méthode retourne une collection de documents classés selon leurs scores calculés. Lucene calcule un score pour chaque document correspondant à une requête.

- **Terme**

Un ***terme*** est l'unité de recherche la plus fondamentale qui soit. Il est composé de deux éléments: le texte du mot et le nom du champ dans lequel ce texte apparait. Les termes sont aussi impliqués dans l'indexation, mais ils sont créés par Lucene de manière interne.

5.4.1Processus de recherche

Pour rechercher l'information stockée dans l'index on exécute le programme de recherche (Searcher) qui va communiquer la requête entrée par l'utilisateur dans la barre de recherche.

Après l'envoie de la requêtes au programme Searcher l'exécution de se dernier passe par plusieurs étapes pour finalement renvoyer à l'utilisateur les éléments pertinents qui répondent à sa requête :

- **Analyse de la requête** :la méthode de QueryPerser fragmente les éléments texte de la requête en termes et traduit Les expressions de larequêteen un type Query et enregistre le résultat de l'analyse dans une variable de même type.

```
QueryParser parser = newMultiFieldQueryParser(Version.LUCENE_30,
                getListField(),new FrenchAnalyzer(Version.LUCENE_30))
Queryquery = parser.parse(Query);
```

- **rechercher dans l'index** :on appelle la Méthode IndexSearcher qui recherche dans l'index qui est stocké dans un répertoire (Directory) et retourne un objet de type TopDocs.

```
IndexSearcher searcher = newIndexSearcher(directory);
TopDocs topDocs = searcher.search(query, maxHits);
```

- **récupération des résultats** : enfin pour récupérer les résultats de l'objet on retourne les élémnet les plus pertinents :
-

```
if(d.getFields().get(j).stringValue().toUpperCase().matches("(.*)"+getQuery().toUpperCase()+
"(.*)")) info += "<br> in field :"+d.getFields().get(j).name()+"<br>";
```

- TopDocsqui renvoi le résultat à l'utilisateur du document contenant l'élément pertinent

```
ScoreDoc[] hits = topDocs.scoreDocs;
intdocId = hits[i].doc;
d = searcher.doc(docId);
d.get("filename");
```

5.4.2Processus d'agrégation

On recherche les éléments les plus pertinents à partir de documents XML et leur agrégation dans un même résultat. Notre objectif est d'assembler automatiquement des éléments pertinents, non-redondants et complémentaires d'un corpus de documents XML qui répondent le mieux au besoin de l'utilisateur formulé à travers une liste des mots clés. Le modèle que nous proposons trouve ses fondements théoriques dans nos algorithmes de recherche. La structure réseau fournit une manière naturelle de représenter les liens entre les éléments du corpus de documents XML et leurs contenus.

Pour l'agrégation des éléments pertinents nous avons élaboré un algorithme qui permet des les agrégés en donnant le nombre d'élément à afficher dans l'agrégat vers la fin de l'opération d'agrégation.

```
TokenStream stream = TokenSources.getTokenStream(fieldName, fieldContents,

analyzer);

SpanScorer scorer = new SpanScorer(query, fieldName,

newCachingTokenFilter(stream));

Fragmenterfragmenter = new SimpleSpanFragmenter(scorer, 100);
```

```
Highlighter highlighter = new Highlighter(scorer);

highlighter.setTextFragmenter(fragmenter);

String[] fragments = highlighter.getBestFragments(stream, fieldContents, 5);
```

5.5Intégration

Arrivé à cette étape le codage de l'application est achevé, nous présentons dans ce qui suit quelques screen shoot de notre application.

5.5.1Présentation de l'application

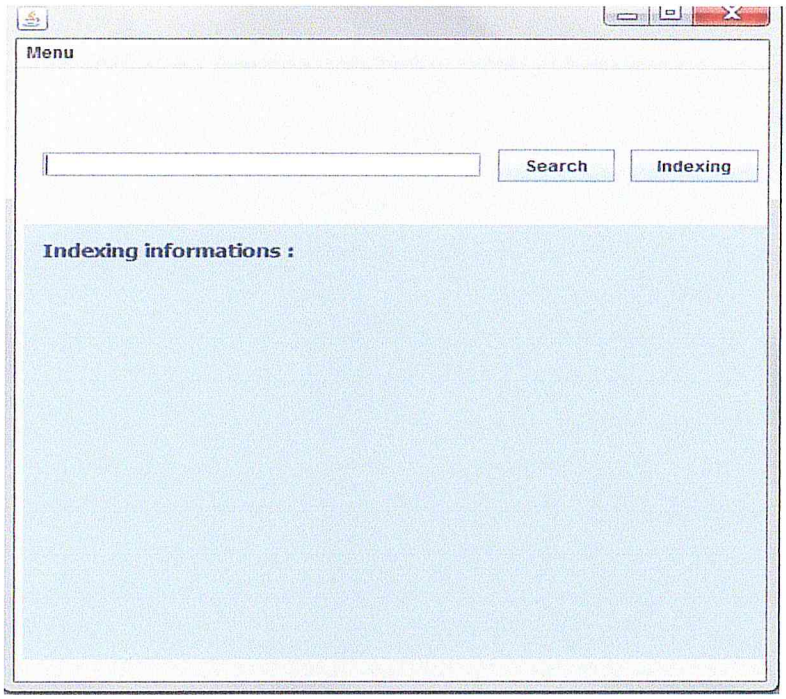


Fig.5.5: Console du projet

- Compilation et exécution

Nous exposons dans cette partie quelques tests d'intégrité qui ont été appliqués sur l'application à fin de vérifier son bon fonctionnement.

Le processus d'indexation

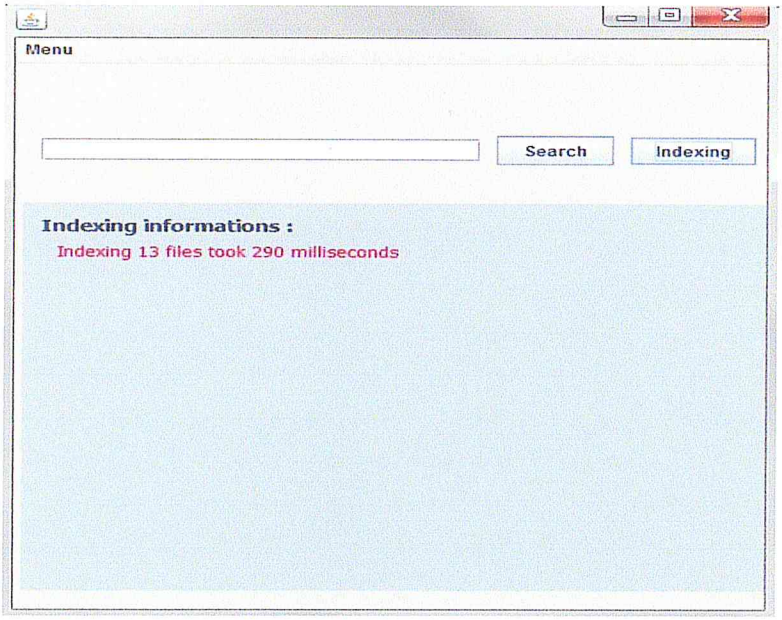


Fig.5.6 : Le processus d'indexation

Le Processus de Recherche

L'utilisateur écrit sa requête puis il clique sur recherche, et il aura comme affichage tous les éléments pertinents qui répondent à sa requête.

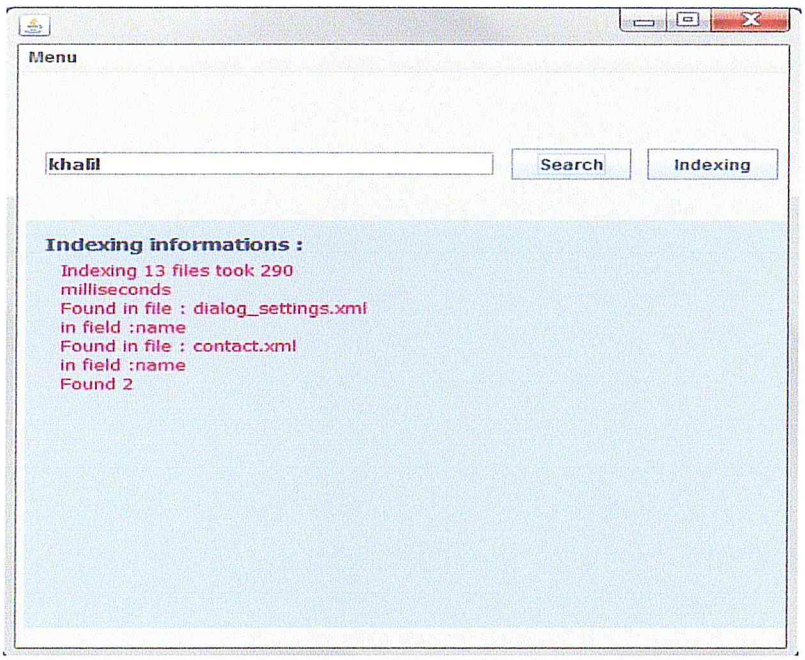


Fig. 5.7 Le Processus de Recherche

5.6. Conclusion

Nous avons abordé dans ce chapitre les phases d'implémentation de notre application.
Nous clôturons, avec ce chapitre, notre cycle de développent avec pour résultat l'application et sa documentation.

Conclusion général

Conclusion Générale

Ce mémoire présente une nouvelle approche pour la RI agrégée dans des documents XML et qui repose sur des algorithmes de recherche puissants, exacts et efficaces en utilisant Lucene, Le un moteur de recherche textuelle Open Source utilisant l'outil de développement JAVA.

Notre modèle montre comment assembler dans un agrégat des éléments XML pertinents, non-redondants et complémentaires afin d'améliorer les résultats de recherche. En effet, les agrégats orientent l'utilisateur plus rapidement pour sélectionner les éléments clés et donnent un aperçu sur les informations disponibles dans le corpus par rapport à la requête.

Ainsi, il semble très intéressant de penser à développer des outils d'évaluation qui évaluent les systèmes de RI agrégée dans but de comparer leurs performances.

Bibliographie

Bibliographie

(Adamson and Boreham, 1974): G. Adamson and J. Boreham. The use of an association measure based on character structure to identify semantically related pairs of words and document titles. Information storage and retrieval.

(AFNOR, 1987) : AFNOR. Les dossiers de la normalisation, ISSN 0297-4827 159 p. ISBN 2-12-484221-8 : 130,85 F HT 1987.

(Barbier, 2005) : Franck Barbier. *UML 2 et MDE*. Dunod, 2005.

(Benferhat, 1999): S. Benferhat, D. Dubois, L. Garcia, H. Prade : Possibilistic logic bases and possibilistic graphs. In Proc. of the 15th Conference on Uncertainty in Artificial Intelligence, 57-64, 1999.

(Bessai, 2005) *: Fatma-Zohra Bessai Mechmache, Mohand Boughanem, ZaiaAlimazighi. Vers un modèle possibiliste pour la recherche d'information dans des documents structurés. 2005.*

(Borgelt, 2000): C. Borgelt, J. Gebhardt, and R. Kruse, Possibilistic graphical models » Computational Intelligence in Data Mining, CISM Courses and Lectures 408, Springer, Wien, 51-68, 2000.

(Boughanem M., Savoy J. 2008). Recherche d'information. État des lieux et perspectives Lavoisier.2008.

(DAV, 1993) : E. Davalo, P. Naïm, Des Réseaux de neurones, Edition Eyrolles, 1993.

(Florescu, 1999):*D. Florescu and D. Kossmann. Storing and querying XML data using an RDBMS. IEEE Data Engineering Bulletin, 22(3) : p.27–34, 1999.*

(Fuhr et al., 2008) : *Fuhr N., Kamps J., Lalmas M., Malik S., Trotman A., "Overview of the INEX 2007 AdHoc Track", p. 1–23, 2008.*

(Hatcher, 2004) Erik Hatcher et Otis Gospodnetić."*Lucene In Action*" par Manning Publications; Decembre 2004.

(Hardt, 2004): Manfred Hardt, Dr. Fabian Theis: "*Suchmaschinenentwickelnmit Apache Lucene*"; Software & Support Verlag, Frankfurt/Main, Germany; Septembre 2004.

(Huang et al., 2008): *Huang Y., Liu Z., Chen Y., « Query biased snippet generation in XML search», SIGMOD'08, p. 315-326, 2008.*

(Kato M. P.et al 2009): Query by analogical example: relational search using web search.

Bibliographie

(Kopliku A., 2009): *Arlind Kopliku. Aggregated search: From information nuggets to aggregated documents. In CORIA 09 RJCRI « Rencontre Jeunes Chercheurs en Recherche d'Information », Toulon, France, 2009.*

(Kopliku A. et al., 2009):*Arlind K., MohandBoughanem, Karen S., « Technical report-IRIT: Aggregatedsearch: potential, issues and evaluation », 2009.*

(MAR, 1991): M. Cord Marilyn, Nelson and W. T. Illengworth, A practical guide to neural nets, Addison-Wesley Publishing company 1991.

(Mignet et al., 2003):*L. Mignet, D. Barbosa, and P. Veltri. The XML web : A first study. In Proceedings ofWWW 2003, Budapest, Hungary, 2003.*

(Michael, 2008): *Michael Shilman. Aggregate documents: making sense of a patchwork of topical documents. InDocEng '08: Proceeding of the eighth ACM symposium on Document engineering, pages 3–7, New York, NY, USA,2008.*

(Murdock et al., 2008):*Murdock V., Lalmas M., « Workshop on aggregated search », SIGIR Forum, p.80-83, 2008.*

(Naffakhi.N et al., 2009) : *Nafakhi N., Boughanem M., Faiz R., « Réseau bayésien pour un modèle de Recherched'Information agrégée dans des documents structurés », 2009.*

(Nielsen, 2003): *Nielsen J., Designing Web Usability, 9th edn, USA, 2003.*

(Oren et al., 1998): *Oren Zamir and Oren Etzioni. Web document clustering: A feasibility demonstration. Pages46–54, 1998.*

(Piwowarski, 2003) : Piwowarski B. Techniques d'apprentissage pour le traitement d'informations structurées: application à la recherche d'information. Thèse de doctorat de l'université de PARIS 6, 2003.

(Porter, 1980): M. F. Porter. An algorithm for suffix stripping. 1980.

(Razan,04) : Razan T. Recherche d'Information Collaborative, Thèse de Doctorat de l'Université Joseph Fourrier, 2004.

(Robertson, 1977): S.E. Robertson. The probability ranking principle in IR. Journal of Documentation.

(Robertson et al., 1994): S. Robertson, S.Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC 3. In Proceedings of the 3rd Text REtrieval Conference (TREC-3).

(Rivadeneira et al., 2003):*W. Rivadeneira and B. B. Bederson. A study of search result clustering interfaces:Comparing textual and zoomable user interfaces. Technical report, University of Maryland, 2003.*

Bibliographie

(Salton, 1970): G. Salton. The SMART retrieval system : Experiments in automatic document processing. Prentice Hall. 1970.

(Salton et al., 1983): Salton G., Fox E. and Wu H., Extended boolean information retrieval. Communications of the ACM.

(Sauvagnat, 2005) : *Karen Sauvagnat. Modèle flexible pour la Recherche d'Information dans des corpus de documents semi structurés. Thèse de doctorat, IRIT, Université Paul Sabatier de Toulouse. 2005.*

(Sévigny, 2002): Martin Sévigny Les documents structures.2002.

(Sekiguti, 2006): Sekiguti Koshi, Gijutsu-Hyohron Co, Ltd *Introduction to Apache Lucene: Construction of Java Open Source Full Text Retrieval Systems*". Mai 2006.

(Spark et al., 1972): K. Sparck-Jones and R. Needham. « Automatic theme classification and retrieval».

(St. Laurent, 2000): *Simon St. Laurent. Introduction a XML. Collection Micro Pro, Osman Eyrolles Multimedia. ISBN: 2-7464-0094-4. 2000.*

(Torjmen et al., 2009):*Torjmen M., Pinel-Sauvagnat K., Boughanem M., « XML multimedia Retrieval: Fromrelevant textual information to relevant multimedia fragments », 31th ECIR, p. 150-161, 2009.*

(Walker et al., 2006): S. Walker, S.E. Robertson, M. Boughanem, G.J.F. Jones, and K. Sparck Jones. Okapi at trec-6: Automatic ad hoc, routing and filtering, 2006.