REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'enseignement supérieur et de la recherche scientifique
Université Mouloud Mammeri de Tizi-Ouzou
X•Θ Λ•Σ X [://:V •X[•Λ[•O
Faculté de génie électrique et informatique
Département d'informatique

Mémoire de fin d'études

En vue de l'Obtention Du Diplôme de Master en informatique

Option : conduite de projets informatiques.

Thème

Conception et réalisation d'une approche d'indexation des documents XML par DataGuide et fichier inverse

Dirigé par :

Mme. AMIROUCHE F.

Réalisé par :

FERROUK Massinissa.

Présidente

Mme. BOUARAB.F

Examinateurs

Mlle ILTACHE.S

Mr. AMIROUCHE.M.N

Remerciements

En préambule à ce mémoire, Nous tenons tout d'abord à remercier Dieu qui nous a donné la force et la patience d'accomplir ce Modeste travail.

Nous tenons à remercier sincèrement Madame AMIROUCHE.F, qui, en tant que notre encadreur, s'est toujours montrée à l'écoute et très disponible tout au long de la réalisation de ce mémoire, ainsi pour l'inspiration, l'aide et le temps qu'elle a bien voulu nous consacrer.

Nous tenons aussi à exprimer toute notre gratitude aux membres du jury pour avoir accepter d'évaluer et de juger notre travail.

Et en fin nous souhaitons adresser nos remerciements les plus sincères aux personnes qui de loin ou de prés nous ont apportés leur aide et qui ont contribué à l'élaboration de ce mémoire.

Table des matières

Chapitre 4 Évaluation et résultat

Listes des figures et des tableaux

Listes des figures et tableaux :

Liste des figures :

Liste des tableaux :

Introduction Générale

Introduction générale

Notre sujet de mémoire a d'abord été attribué dans le cadre d'un trinôme (DJELOUAH Chafik, FENEK Aghiles et FERROUK Massinissa).

La recherche bibliographique, première étape de ce travail a été réalisée dans ce contexte. Puis suite au remaniement du trinôme dicté par une note administrative le mémoire a été subdivisé comme suit :

- **Thème 1** : qui concerne une conception et réalisation d'une approche d'indexation des documents XML par DataGuide et fichier inverse réalisé en monôme par FERROUK Massinissa.
- **Thème 2** : qui concerne une conception et réalisation d'un système de recherche d'information semi-structuré conceptuel réalisé en binôme par DJELOUAH Chafik et FENEK Aghiles.

L'état de l'art initialement réalisé à trois a été rajouté dans les deux mémoires résultant du remaniement.

Notre travail dans ce mémoire se situe dans le contexte de la recherche d'information(RI) dans les documents semi-structurés (RIS) de type XML. La RI traditionnelle traite des documents non-structurés où l'unité de recherche pertinente correspond à un document dans sa globalité contrairement à la RIS où l'unité de recherche est variable et correspond à un élément (paragraphe, section, titre…) dans le document. La granularité de la recherche XML est ainsi plus fine qu'en RI traditionnelle.

La prise en compte de la structure a amené de nouvelles problématiques et de nouveaux défis à différents niveaux de la RI :

- Au niveau de l'indexation/appariement : l'indexation des documents XML doit tenir compte de la structure et du contenu des documents XML tout en gérant le lien entre les deux.

- Au niveau de l'interrogation : les langages des requêtes permettent à l'utilisateur d'interroger les documents semi-structurés. Ces langage de requête doivent supporter à la fois des contraintes portant sur la structure et le contenu.

Nous nous intéressons dans notre travail au problème de l'indexation des documents semi-structurés et plus particulièrement à la représentation des index. Les structures de données utilisées en RI classique pour la représentation des index sont le fichier inverse et le fichier de signature. Ces structures de données sont utilisées pour représenter l'index de contenu, elles ne prennent pas en charge l'information structurelle. Pour l'indexation de la structure des documents, de nouvelles structures d'index on tété proposées dont la plus performante est sans doute le DataGuide. Le DataGuide est une représentation compacte de la structure des documents d'une collection.

Les approches d'indexation des documents semi-structurés sont ainsi basées sur l'utilisation conjointe d'un index de contenu (généralement le fichier inverse) et d'un index de structure (le DataGuide). Néanmoins les approches existantes ne résolvent que partiellement la problématique, car elles ne conservent pas un lien fort entre l'index de la structure et celui du contenu, ce qui dégrade la performance du SRI.

Dans notre travail, nous proposons une approche d'indexation de documents XML basée sur l'utilisation conjointe du fichier inverse et du DataGuide. Cette approche préserve un lien fort entre l'index de la structure est celui du contenu par un lien virtuel crée au moment de l'indexation via une signature virtuelle. Cette approche est fondée sur trois étapes :

- Une première étape qui permet l'indexation de la structure des documents avec un DataGuide.
- La seconde étape permet de construire le fichier inverse qui contient l'index de contenu.
- Une troisième étape permettant de relier l'index de contenue à celui de la structure avec une signature virtuelle.

Notre mémoire est articulé sur 4 chapitres principaux :

L'objectif du premier chapitre est de présenter les méthodes, modèles et concepts fondamentaux de la RI « classique ».

Dans le second chapitre nous présentons les modèles et algorithmes fondamentaux utilisés en recherche d'information semi-structurée.

Le troisième chapitre introduit notre une approche d'indexation pour la recherche des documents semi-structurés de type XML basée sur l'utilisation conjointe d'un index de structure (la DataGuide) et d'un index de contenu (le fichier inverse), tout en préservant le lien entre les deux via une signature virtuelle.

Le quatrième chapitre présente les résultats de l'évaluation de notre approche sur la collection INEX 2009.

Chapitre 1
Recherche d’information classique

I.1Introduction

La recherche d'information a pour objectif de retrouver dans une collection donnée, l'ensemble des documents pertinents pour un besoin informationnel, exprimé sous forme d'une requête.

Un Système de Recherche d'Information (SRI) est un ensemble de programmes informatiques qui a pour but de retrouver dans une **collection de documents**, les **documents** qui sont pertinents pour un besoin informationnel de l'utilisateur exprimés sous forme d'une **requête**.

Cette définition fait ressortir les éléments clés suivants :

1. **Document :** On appelle un document toute unité qui peut constituer une réponse à une requête d'utilisateur et il peut être présenté sous plusieurs formes, un texte, un morceau de texte, une image, une bande vidéo, une page web…
2. **Collection de documents** : Ensemble de documents
3. **Requête :** Une requête exprime le besoin d'information d'un utilisateur.
4. **La pertinence système :** Elle représente la relation entre la requête et l'information. « Elle définit une correspondance entre un document et une requête, ou encore une mesure d'normativité du document à la requête » (Boughanem et al, 08)
5. **La pertinence utilisateur :** c'est une mesure subjective qui représente la satisfaction de l'utilisateur vis-à-vis des documents retournés.

   L'objectif d'un SRI est de faire correspondre la pertinence système à la pertinence utilisateur.

I.2 Le processus de recherche d'information

Le processus ou le modèle général de la recherche d'information a pour but la mise en relation des différents acteurs de la recherche d'information : collection documentaire (corpus), Utilisateur (besoin informationnel) et le Système de Recherche d'Information qui doit pouvoir traiter une grande masse d'informations de façon rapide et pertinente.

Le rôle d'un SRI est retrouver les documents pertinents pour un besoin informationnel de l'utilisateur formellement exprimé par une requête. Pour cela, le SRI indexe les documents et la requête dont l'objectif de créer leurs représentations internes. Ces représentations internes sont ensuite appariées selon un modèle de recherche, et les documents les plus similaires à la requête sont retournés triés par ordre de pertinence.
Un SRI peut en outre intégrer un processus de reformulation de requête permettant d'améliorer les résultats de la recherche.

Le fonctionnement général d'un SRI est donné au travers du processus de recherche communément appelé processus en U [Belkin et al., 92], présenté en figure 1.1.

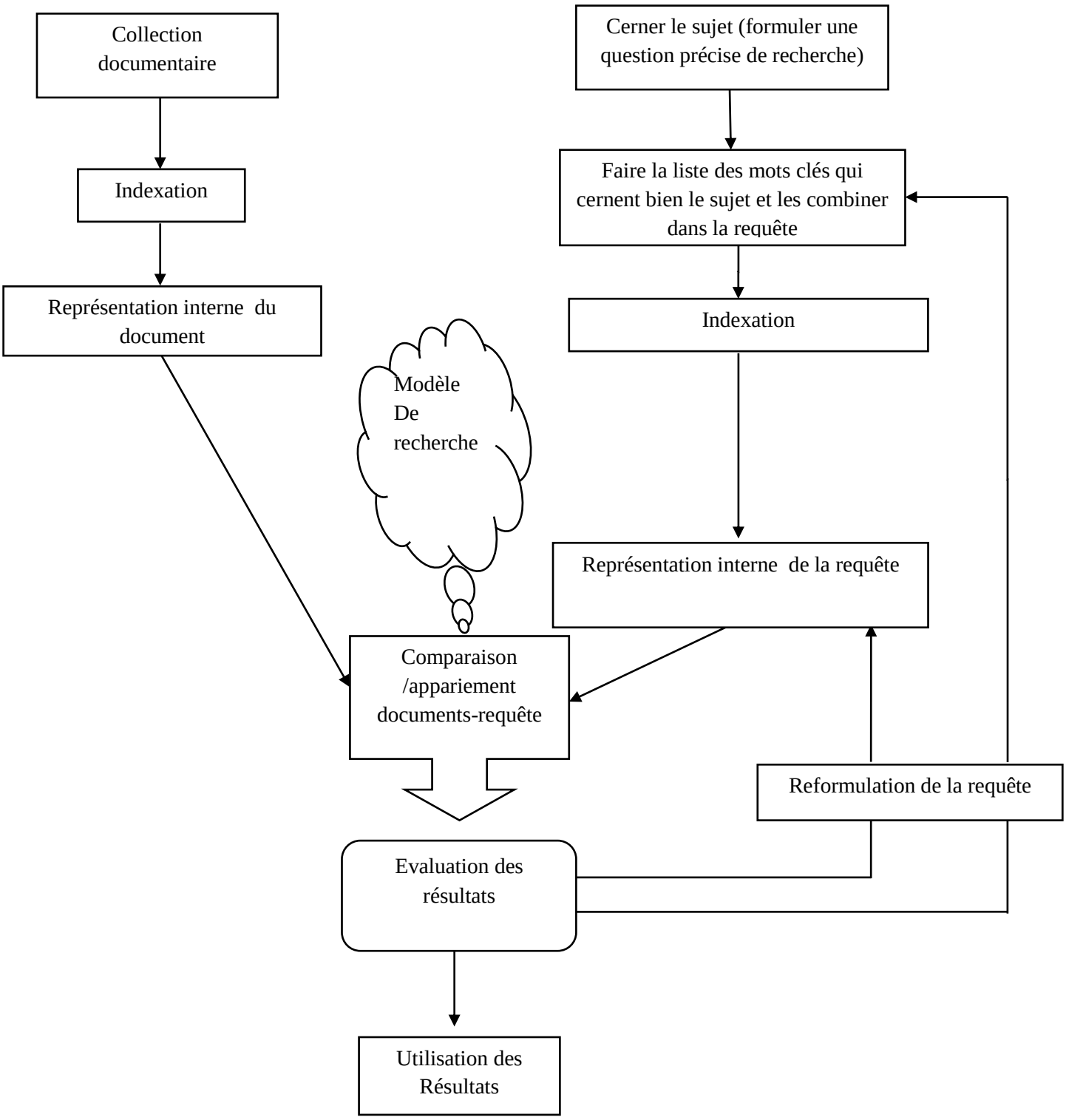


FIGURE 1.1 : Processus en U de la RI

I.2.1 L'indexation :

L'indexation est une phase fondamentale et indissociable d'un SRI. De sa qualité dépend la performance du SRI. L'indexation a pour but de construire l'ensemble de termes représentatifs du contenu de la requête et de chaque document de la collection. Ces termes, dits termes d'index, jouent un rôle très important dans la recherche d'information puisqu' ils déterminent avec quels mots on peut retrouver un document.

L'ensemble des termes d'index constituent le langage d'indexation. Ce langage peut être :

- ✓ **Un langage libre :** il est construit à partir des termes extraits du document analysé.
- ✓ **Un langage d'indexation contrôlé** : il est construit à partir d'un ensemble de termes préalablement définis et organisés dans un thésaurus ; l'indexation se fait seulement avec les termes de cette terminologie.

I. 2.1.1 Approches d'indexation :

L'indexation peut être manuelle, automatique ou semi-automatique :

- ✓ **L'indexation manuelle** :
  C'est une opération humaine où chaque document est analysé par un spécialiste ou un documentaliste.

- ✓ **L'indexation automatique** :
  C'est la technique la plus utilisée car la plus adaptée aux corpus volumineux. Le processus d'indexation est entièrement informatisé. Aucune intervention humaine n'est requise dans le choix des termes d'index.

- ✓ **L'indexation semi-automatique** :
  C'est une indexation mixte combinant les deux approches précédentes. Les termes d'index sont d'abord extraits automatiquement, puis le choix final des termes est réalisé par les spécialistes ou documentalistes.

I.2.1.2Indexation automatique :

L'indexation automatique est fondée sur l'analyse des documents en vue de l'extraction des termes représentatifs de leur contenu informationnel. L'indexation automatique repose sur les étapes suivantes :

a- L'extraction des termes d'indexation : repose sur une analyse linguistique du texte du document. Elle comprend les étapes suivantes :

- ✓ Analyse lexicale : l'analyse lexicale est un processus qui permet de transformer un texte du document en un ensemble de termes.
- ✓ Elimination des mots vides : Les mots représentatifs sont gardés et les mots vides (mots outils de la langue comme les propositions, les articles, les pronoms…, mots trop fréquents, mots rares) sont éliminés.
- ✓ Normalisation des termes d'index basée sur deux approches:

- **La lemmatisation :** La lemmatisation d'un mot consiste à transformer le mot en son lemme. Le lemme est la racine du mot telle que définie dans la langue naturelle. L'algorithme le plus populaire de lemmatisation pour la langue anglaise est l'algorithme de Porter.

Example: Regroupement par **lemme**

am, are, is➔be

car, cars, car's, cars'➔ car

Exemple de règles de fonctionnement de l'algorithme de Porter :

SSES ===>SS
caresses ===> caress
IES ===> I
ponies ===>poni
SS ===>SS
caress ===> caress
S===>
cats ===> cat

- **La troncature :** Construit le radical d'un mot en le coupant à partir d'une position donnée. De cette manière, on réduit les variations morphologiques des mots.

  Par exemple : le terme **Psycholog** est obtenu par troncature à 9 caractères des mots suivants qu'il représente **:**

  - **psychologie ;**
  - **psychologies ;**
  - **psychology ;**
  - **psychologue(s) ;**
  - **psychological ;**
  - **psychologist(s)…**

b- La pondération des termes d'index:

La pondération consiste à donner un poids d'importance W_{ij} à chaque terme t_i d'un document d_j. Les approches de pondération se basent le plus souvent sur deux facteurs :

- un facteur **de pondération locale** qui exprime l'importance du terme t_i dans le document d_i. Ce facteur est fonction de sa fréquence d'occurrences tf_{ij} dans le document.

- un facteur **de pondération globale** qui exprime son pouvoir de discrimination sur l'ensemble des documents de la collection. Ce facteur est fonction de sa fréquence documentaire inverse $\boldsymbol{idf_j}$ dans tous les documents de la collection.

Plusieurs formules existent et sont basées sur le schéma combiné suivant:

$$\mathbf{W_{ij}} = = \boldsymbol{tf_{ij}} * \boldsymbol{idf_j}$$

Formule 1.1

Où

$\boldsymbol{tf_{ij}}$: est la fréquence du terme $\mathbf{t_j}$ dans le document

$\boldsymbol{dfj_i}$. : est fréquence documentaire du terme (ie. Le nombre de documents de la collection contenant ce terme)

$\boldsymbol{idf_j}$: est la fréquence documentaire inverse du terme $\mathbf{t_j}$.

I.2.2 Les modèles de la recherche d'information :

Un modèle en recherche d'information a pour but de fournir une formalisation du processus de recherche d'information. A partir des termes issus de la phase d'indexation, le modèle remplit les rôles suivants :

- représentation interne d'un document et représentation interne de la requête.

- Définir une méthode de comparaison entre ces deux représentation afin de déterminer la similarité document /requête.

On distingue principalement trois classes de modèles de recherche d'information :

- **Les modèles basés sur la théorie des ensembles** : dont **le modèle booléen** qui est le modèle le plus utilisé dans la recherche d'information.

- **Les modèles algébriques :** dont le **modèle vectoriel** qui considère que les documents et la requête font partie d'un même espace vectoriel**.**

- **Les modèles probabilistes** qui se basent sur les probabilités pour estimer la pertinence d'un document vis-à-vis d'une requête.

La figure 1.2 représente la taxonomie des modèles de la recherche d'information :

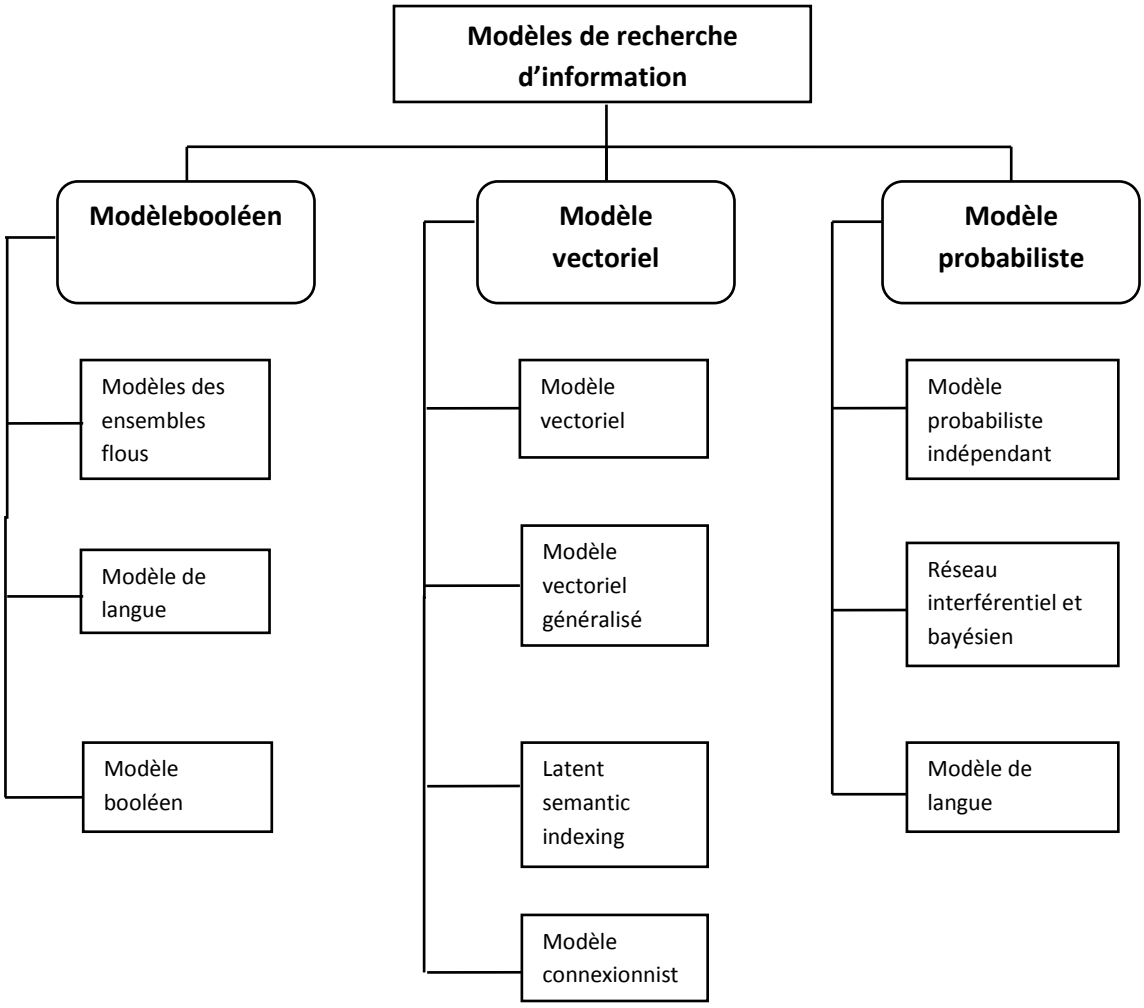
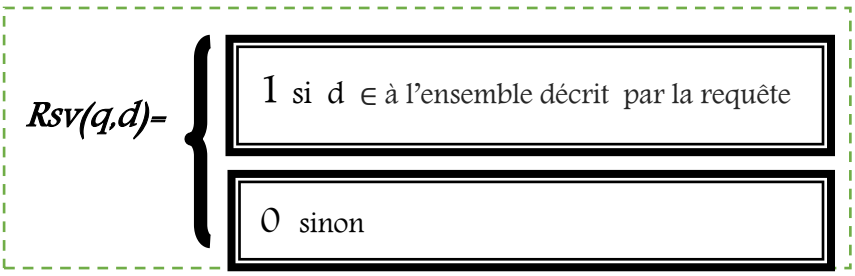


FIGURE 1.2 : Taxonomie des modèles en RI

I. 2.2.1 Le modèle booléen :

Le modèle booléen est basé sur la théorie des ensembles et l'algèbre de Boole. Dans le modèle de base, le document est représenté par un ensemble de termes (non pondérés), et la requête est définie par un ensemble de mots clés reliés par des opérateurs logique (AND, OR, NOT). Ce modèle considère que si les termes de la requête sont présents dans le document alors le document est pertinent pour la requête. Son score de pertinence pour la requête (RSV(q,d)) prend alors la valeur 1. Dans le cas contraire, le document est considéré comme non pertinent, et son score de pertinence prend la valeur 0.



L'appariement requête/document est strict et se base sur des opérations ensemblistes selon les règles suivantes :

RSV (d, t_i) = 1 si $t_i \in d$, 0 sinon

RSV (d, t_i AND t_j) = 1 si ($t_i \in d$) $\wedge$ ($t_j \in d$) , 0 sinon

RSV (d, t_i OR t_j) = 1 si ($t_i \in d$) $\vee$ ($t_j \in d$) , 0 sinon

RSV (d, NOT t_i) = 1 si $t_i \notin d$, 0 sinon

Bien que ce modèle soit simple à mettre en œuvre, il présente des inconvénients majeurs:

- ✓ les termes ne sont pas ordonnables et sont considérés de même importance tant dans le document qua dans la requête ;
- ✓ On ne peut classer les documents que dans deux catégories (pertinents ou non pertinents)
- ✓ Les expressions booléennes ne sont pas accessibles à un large public.

Remarque :

Pour pallier aux inconvénients du modèle booléen de base, plusieurs extensions ont été proposées [le modèle p-norme, Salton et al., 1983], qui visent à tenir compte des poids des termes dans les documents et dans la requête, ce qui permet l'ordonnancement des documents retrouvés par ordre de pertinence (valeur approchée).

I.2.2.2 Les modèles vectoriels :

I.2.2.2.1 Le Modèle vectoriel de base :

C 'est le modèle le plus populaire et le plus utilisé dans la recherche d'information. Dans ce modèle, un document $\mathbf{d_i}$ est représenté par un vecteur de poids $\mathbf{w_{i.j}}$de dimension **n** composé de tous les termes d'index.

$$\mathbf{d_i=(w_{i.1},w_{i.2},\ldots., w_{i.n})\ pour\ i = 1, 2, \ldots, m}$$

Où

- $\mathbf{w_{i.j}}$ est le poids du terme $\mathbf{t_i}$ dans le document $\mathbf{d_i}$;
- **M** est le nombre de documents de la collection;
- **n** est le nombre de termes d'index dans la collection.

Une requête est aussi représentée par un vecteur de poids $wQ_{i.j}$ défini dans le même espace vectoriel que le document

$$\mathbf{Q = (wQ_1 , wQ_2, \ldots, wQ_n)}$$

Où

- $\mathbf{wQ_j}$ est le poids du terme $\mathbf{t_j}$ dans la requête **Q** (ce poids peut être soit une forme de **tf*idf**, soit un poids attribué manuellement par l'utilisateur)

La pertinence du document $\mathbf{d_i}$ pour la requête **Q**, est définie comme le degré de corrélation des vecteurs documents/requête. Cette corrélation peut être exprimée par l'une des mesures suivantes :

- ✓ **Le produit scalaire :**

$$\mathbf{Sim\ (d_i , Q) = \left(\sum_{i=1}^{n} w_{Qj} * W_{ij} \right)}$$

Formule 1.2

- ✓ **La mesure de cosinus :**

$$\mathbf{Sim\ (d_i,Q)=\frac{\sum_{i=1}^{n} WQj*Wij}{(\sum_{i=0}^{n} W^2Qj)^{1/2}*(\sum_{j=1}^{n} W^2_{ij})^{1/2}}}$$

Formule 1.3

- ✓ **La mesure de Dice :**

$$\mathbf{Sim\ (d_i , Q) = \frac{2*\sum_{i=1}^{n} WQj*Wij}{\sum_{j=1}^{n} W_{QJ}^2 +\sum_{j=1}^{n} W_{ij}^2}}$$

Formule 1.4

- ✓ **La mesure de Jacard :**

$$\textbf{Sim } (d_i, Q) = \frac{\sum_{j=1}^{n} W_{ij} * w_{Qj}}{\sum_{j=1}^{n} W_{QJ}^2 + \sum_{j=1}^{n} W_{ij}^2 - \sum_{j=1}^{n} W_{ij} * W_{Qj}}$$

Formule 1.5

- ✓ **Le coefficient de superposition (overlap) :**

$$\textbf{Sim } (d_i, Q) = \frac{2 * \sum_{i=1}^{n} WQj * Wij}{\min_i (\sum_{j=1}^{n} W_{QJ}^2 , \sum_{j=1}^{n} W_{ij}^2)}$$

Formule 1.6

I.2.2.2.2 Le modèle connexionniste :

L'idée de base est que la RI est un processus associatif qui peut être représenté par un processus d'activation des réseaux de neurones. La notion de réseaux est très intéressante pour la représentation des différentes relations et associations qui existent entre les termes, les documents et les (termes et documents).

Le modèle connexionniste utilise le fondement de réseaux de neurones pour la modélisation des unités textuelles et la mise en œuvre du processus de la recherche d'information. Deux modèles théoriques ont été utilisés:

- ✓ Les modèles à auto-organisation [Lin et al., 91] :permettent à partir de la description des documents, d'en réaliser une classification par l'apprentissage du réseau de neurones.
- ✓ Les modèles a couches : Les SRI se basent sur un modèle connexionniste à couches [Kwok, 89; Belew, 89; Boughanem, 92; Mothe, 94] et sont représentés par un minimum de trois couches de neurones interconnectées :
    - la couche requête (Q) ;
    - la couche termes (T) ;
    - la couche documents (D) ;

Le mécanisme de recherche est basé sur une activation initiale de neurones termes induite par une requête, et qui se propage vers les documents à travers les réseaux. Les documents qui ont des termes en commun avec la requête recevront une entrée **In(d_i)** et le modèle calculera sa sortie **Out(d_i)**. Les valeurs de sortie des différents documents correspondent à leurs degrés de pertinence pour la requête donnée.

I.2.2.3 Les modèles probabilistes :

I.2.2.3.1 Le modèle probabiliste de base :

Le premier modèle probabiliste a été proposé par **Maron** et **Kuhns [Maron et al., 60]** au début des années 60. Le modèle probabiliste consiste à présenter les résultats d'un SRI dans un ordre basé sur la probabilité de pertinence d'un document vis-à-vis d'une requête.

L'idée de base dans un modèle probabiliste est de tenter de déterminer la probabilité P(R|D) que le document D sélectionné soit pertinent pour la requête Q et la probabilité P(NR|D) que le document soit non pertinent pour Q.

Le score de pertinence d'un document (ou**RSV(d ,Q)**) est donné, d'après le théorème de Bayes et après simplification, par :

$$\mathbf{RSV(d, Q)} = \frac{p(R/di)}{p(NR/di)} \approx \frac{p(di/R)}{p(di/NR)}$$

Formule 1.7

Où:

$P(d_i/R)$ est la probabilité que d_i appartienne à l'ensemble des documents pertinents
$P(d_i/NR)$ est la probabilité que d_i appartienne à l'ensemble des documents non pertinents

I.2.2.3.2 Modèle de langue :

Le modèle de langue considère que la pertinence d'un document vis-à-vis d'une requête est en rapport avec la probabilité que la requête puisse être générée par le document. Un document est pertinent si et seulement si la requête utilisateur est en ressemblance avec celle inférée par le document :

$$\mathbf{RSV(d,Q) = P(Q = (t_1,t_2, \ldots t_n)/Md)} = \prod_{i=1}^{n} p(t_i/d)$$

Formule 1.8

$P(t_i/d)$ peut être estimé en se basant sur l'estimation maximale de vraisemblance comme suit :

$$\mathbf{P\ (ti\ /d)} = \frac{\mathbf{tf(t_i/d)}}{\sum_t \mathbf{tf(t/d)}}$$

Formule 1.9

I.2.3 La reformulation de requête :

Vu la croissance phénoménale des bases documentaires l'utilisateur trouve des difficultés à formuler une question précise de la recherche. Afin de correspondre la pertinence utilisateur et la pertinence système, une étape de reformulation de requête est souvent utilisée.

La reformulation de requête signifie qu'à partir d'une requête initiale formulée par l'utilisateur, il puisse construire une nouvelle requête qui répond mieux à ses besoins informationnels.

Il existe deux principales méthodes de reformulation de requêtes: la méthode locale et la méthode globale.

I.2.3.1 La méthode locale :

Les méthodes locales s'appuient sur la technique de réinjection de pertinence ou *relevance feedback* [Buckley et al., 94; Harman,92; Robertson et al., 97; Rocchio, 71] . L'idée de base est de faire participer l'utilisateur dans le processus de reformulation.

- L'utilisateur formule sa requête.
- Le système renvoie un premier ensemble de résultat de recherche.
- L'utilisateur marque quelques documents retournés comme pertinents ou non pertinents.
- Le système calcule une meilleure représentation du besoin en l'information sur la base de rétroaction utilisateur.
- Le système visualise un ensemble révisé de résultats de la recherche

Le processus de réinjection de pertinence peut passer par une ou plusieurs itérations de ce type. La reformulation de la nouvelle requête se fait à l'aide de la pertinence utilisateur, La nouvelle requête $\mathbf{Q_m}$est obtenue à partir de la requête initiale **Q0** en appliquant un algorithme spécifique de réinjection de pertinence.

Dans le modèle vectoriel, l'algorithme de reformulation de requête le plus utilisé est l'algorithme de Rocchio [Salton et al., 1983; Salton, 1989] . Dans cet algorithme, la nouvelle requête **Qm** est construite comme suit :

$$\mathbf{Qm} = \alpha \mathbf{Q0} + \beta \frac{1}{|\mathbf{R}|} \sum_{\mathbf{d_p} \in \mathbf{R}} \mathbf{d_p} - \gamma \frac{1}{|\mathbf{NR}|} \sum_{\mathbf{d_{np}} \epsilon \mathbf{NR}} \mathbf{d_{np}}$$

Formule 1.10

Où :
Qm : la requête modifiée
Q0 : la requête initiale

dp : (respectivement **dnp**) est un vecteur associé à un document pertinent (respectivement non pertinent)
R : est l'ensemble des documents pertinents.
NR : est l'ensemble des documents non pertinents.
α, β et γ étant des constantes telles que **α+β+γ=1**
Les paramètres **α, β et γ** sont choisis en fonction de l'importance qu'on souhaite donner à la requête initiale.

I.2.3.2 Méthode globale :

Les méthodes globales se basent sur l'expansion de requête. La forme la plus commune d'expansion de requête est l'analyse globale en utilisant un thesaurus [Qiu, 93] ou une ontologie [Mandala et al., 91; Voorhees, 94; Navigli et al., 03; Moldovan et al., 00; Bazizet al., 03a ; Baziz et al., 03b]. Pour chaque terme t, la requête peut être automatiquement étendue avec des mots du thésaurus synonymes ou liés au terme t. Le système peut ainsi apparier la requête à des documents pertinents qui ne contiennent aucun des mots de la requête originale.

I.3 Évaluation d'un SRI :

L'évaluation d'un SRI est une étape très importante pour sa validation. Elle permet de définir les caractéristiques du système en termes de qualité de service et de facilité d'utilisation selon les critères suivants:

- le temps de réponse,
- la présentation des résultats,
- l'effort fourni par l'utilisateur pour retrouver parmi les documents retournés ceux qui sont pertinents,
- le taux de rappel du système
- le taux de la précision du système.

Le temps de réponse, la représentation des résultats et l'effort fourni par l'utilisateur sont des mesures de la qualité de service rendu à l'utilisateur tandis que le rappel et la précision essentiels aux modèles de recherche couvrent quant à eux la performance du système.

L'évaluation d'un SRI consiste principalement à mesurer ces performances sur la base d'une **collection de test** contrôlée et des **métriques d'évaluation** standards définis selon des critères d'efficacité.

I.3.1 Collections de référence - Un exemple : les collections TREC :

Une collection de test (ou collection de référence) comprend un ensemble de documents à indexer sur lesquels le système sera évalué, une liste de requêtes prédéfinies et des jugements de pertinence manuellement établies par des assesseurs humains pour chaque requête.

De nombreuses collections de référence existent en RI. Elles sont principalement mises en œuvre dans le cadre de campagnes d'évaluation, dont les principales sont :

- **La campagne CLEF :** lancée en 2000, cette campagne a pour objectif de promouvoir la recherche et le développement dans le domaine de la recherche d'information multilingue.
- **La campagne Amaryllis :** est une version française de TREC de 1996 à 1999 et sous taches de CLEF en 2002
- **La campagne INEX :** Lancée en 2002, son objectif principal est de promouvoir l'évaluation de la recherche dans les documents semi-structurés en fournissant de grandes collections de test de type XML.
- **La campagne TREC** (textretrieval conférence) : constitue le projet le plus ambitieux d'évaluation des SRI. La campagne TREC est une campagne d'évaluation annuelle depuis 1992. Elle vise à explorer de nouveaux domaines de recherche et de démontrer la robustesse des méthodologies de recherche existantes. Chaque année, la campagne TREC lance de nouvelles tâches (ou pistes) correspondant aux centres d'intérêts actualisés des chercheurs de RI. Parmi ces tâches, on distingue :
    1. La tâche *spoken document retrieval*
    2. La tâche *question answering*
    3. La tâche Interactive
    4. **La tâche *Ad Hoc* :** c'est la tâche classique de RI. Elle vise à évaluer les performances d'un SRI sur des ensembles statiques de documents.
    5. **…**

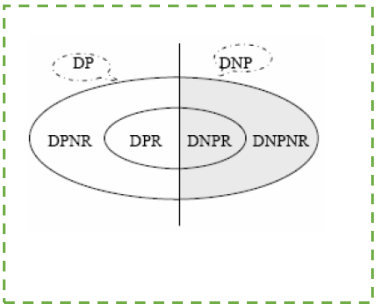
I.3.2 Les mesures d'évaluation d'un SRI :

Pour évaluer la performance d'un SRI, deux principales mesures sont utilisées

- ✓ **la précision :** détermine l'aptitude d'un SRI à rejeter les documents non pertinents vis à vis à d'une requête utilisateur
- ✓ **le rappel :** exprime la capacité d'un SRI à sélectionner tous les documents pertinents vis à vis de cette requête.

Etant donnée une requête Q, les documents de la collection peuvent être globalement classifiés en fonction de leur rapport à la requête en :

- ensemble de documents pertinents DP
- ensemble de documents pertinents retrouvés DPR
- ensemble de documents pertinents non retrouvés DPNR
- ensemble de documents non pertinents DNP
- ensemble de documents non pertinents retrouvés DNPNR
- ensemble de documents non pertinents non retrouvés



Formellement :

$$\text{Rappel} = \frac{|DRP|}{|DP|} \qquad \text{Précision} = \frac{|DPR|}{|DPR \cup DNPR|}$$

Formule 1.11

Les mesures de rappel et de précision utilisées seules ne sont pas de bons indicateurs de la performance d'un SRI Par ailleurs, dans certaines applications, le rappel peut être plus intéressant que la précision. Comment trouver un compromis entre le rappel et la précision ? Plusieurs approches proposées dont : Agrégation du rappel et de la précision dans une seule mesure : le F-score [VanRijbergen, 79].

La F-mesure permet d'agréger le rappel et la précision dans une mesure unique :

$$F = \frac{1}{a*\frac{1}{p}+(1-a)*\frac{1}{R}} \quad \text{où a €[0..1]}$$

Formule 1.12

Apres développement, on obtient la forme communément utilisée suivante

$$F_\beta = \frac{(\beta^2+1)*P*R}{\beta^2*P+R} \quad \text{Où} \quad \beta=\frac{1-a}{a}$$

Formule 1.13

La F-mesure par défaut est donnée comme suit :

$$F_1 = \frac{2*P*R}{P+R}$$

Formule 1.14

- Les mesures de rappel, précision et F-score sont des mesures basées-ensembles :
- Elles permettent d'évaluer des ensembles non ordonnés de résultats.
- On parle de mesures d'évaluation non ordonnées.

D'autres mesures d'évaluation ordonnée existent : De nombreuses mesures d'évaluation permettent d'agréger le rappel et la précision en une valeur unique, parmi lesquelles:

- La précision moyenne Pmoy,
- La MAP,
- La R-précision(ou précision exacte).

La précision Moyenne :

L'idée est de générer une valeur unique de ranking en moyennant les valeurs de précision obtenues après chaque document pertinent observé.

MAP :

La MAP est la moyenne des précisions moyennes (Pmoy) obtenues sur l'ensemble des requêtes à chaque fois qu'un document pertinent est retrouvé :

$$MAP = \frac{\sum_{q \in Q} P_{moy}(a)}{|Q|}$$

Formule 1.15

Q étant l'ensemble des requêtes.

R-précision :

La R-précision est un bon paramètre pour observer le comportement d'un système pour chaque requête individuellement.

La R-précision moyenne calculée sur toutes les requêtes n'a pas d'intérêt.

L'idée est de générer une valeur de ranking unique en calculant la précision au rang R, où R est le nombre de documents pertinents pour la requête courante.

$$R - Prec = P@R = \frac{|DPR|}{R}$$

Formule 1.16

I.3.3 Protocole d'évaluation TREC :

Pratiquement, pour évaluer un SRI, les participants à la campagne TREC doivent suivre le protocole suivant : Pour chaque requête de la collection de test fourni, les 1000 premiers documents restitués par le système sont examinées et les précisions à x points (notées P@ x), sont calculées à différents points (à 5, 10, 15, 30, 100 et 1000 premiers documents restitués). La précision exacte découle de ces précisions. Puis, une précision moyenne MAP est calculée pour chaque requête. Il s'agit de la moyenne des précisions de chaque document pertinent pour cette requête. La précision d'un document est la précision à x, tel que x est le rang de ce document dans l'ensemble des documents pertinents retrouvés.

Finalement, les précisions moyennes pour l'ensemble des requêtes sont calculées permettant d'obtenir une mesure de la performance globale du système.

I.4 Conclusion :

Dans ce premier chapitre, nous avons présenté les méthodes, modèles et concepts fondamentaux de la RI « classique ».

Les modèles décrits dans ce chapitre fonctionnent généralement sur tous les formats (structurés, non structurés et balisés, non balisés), mais ils ne permettent d'exploiter que le contenu textuel du document. Or, avec l'avènement du web, de plus en plus de documents sont structurés ou semi-structurés de format XML. La structure apportant des informations supplémentaires qu'il est utile d'exploiter en vue d'affiner la recherche. Il devient donc impératif de réfléchir à des SRI capables de tenir compte de la structure en plus du contenu pour une recherche efficace. Dans le chapitre suivant, nous présentons des modèles de RI traitant les documents comportant du contenu et de la structure.

Chapitre 2
Recherche d’information semi-structurée

II.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 ou semi-structurés, du SGML au HTML et autre XML qui a vu son importance augmenter grâce à l'expansion d'Internet.

La communauté de la RI s'intéresse actuellement à cette évolution, car elle voit dans la structure un moyen intéressant permettant d'affiner aussi bien la représentation des documents que la notion d'unité d'information, qui était jusque-là liée au document entier. Mais la coexistence de l'information structurelle et de l'information de contenu dans ce type de documents soulève de nouvelles problématiques parmi lesquelles :

- L'indexation des documents et la pondération des termes qui doit prendre en considération l'information structurelle.
- L'interrogation des documents qui doit permettre d'exprimer des besoins diversifiés portant sur le contenu et/ou sur la structure
- Le modèle de recherche où lors de l'appariement on doit aussi prendre en compte les conditions structurelles…

Pour répondre à ces nouvelles problématique, la communauté en RI a du parfois adapter les techniques de la RI Classique (vectoriel, probabiliste, de langue, etc.), et d'autres fois proposer de nouvelles.

Dans ce chapitre, nous présentons la notion documents structurés et documents semi-structurés. Nous nous intéressons en particulier aux documents XML objets de notre étude. Nous présentons ensuite les problématiques rencontrées en recherche d'information semi-structurée, ainsi que les techniques et modèles proposés par la communauté de la RI pour solutionner ces dernières.

II.2 Document et Structure

Les documents structurés et semi-structurés sont des documents dans lesquels le contenu est organisé par une suite de balises hiérarchiquement imbriquées définissant sa structure logique.

Une balise encadre une zone documentaire appelée élément qui peut contenir elle aussi d'autres éléments. Au niveau le plus bas, un élément encadre une zone de texte élémentaire dite atome. Un atome peut être du contenu textuel ou multimédia. Il est non structuré. La balise est identifiée par un nom ou label qui n'est pas spécifique à un document donné mais peut être attribué à plusieurs autres documents.

Il y a une différence entre documents semi-structurés et documents structurés. Les premiers sont hétérogènes et peuvent contenir des 'contenus mixtes' (un élément contenant du contenu textuel et un ou plusieurs autres éléments), alors que les seconds ne contiennent pas de contenu mixte et ont une structure assez régulière. Les documents XML, auxquels nous nous intéressons dans le cadre de notre travail, sont des documents semi-structurés.

II.3 Les documents XML

II.3.1 Le langage XML

XML (*Extensible Markup Langage*) ou (langage à balises extensible) est un standard mis en place par le W3C, qui comporte les fonctionnalités du SGML mais d'une façon réduite afin de le rendre utilisable sur le web. Le langage XML est un format de description de données, permettant de structurer le contenu des documents en les marquant par des balises. Le langage XML est dit générique car le choix des noms des balises et des attributs ainsi que leur organisation sont laissés au libre choix du créateur.

II.3.2 Structure des documents XML :

La structure du document XML décrit les différentes parties qui le composent. Tout document XML est composé de 3 parties :

- un Prologue, dont la présence est facultative mais conseillée : il contiendra un certain nombre de déclarations.
- un arbre d'éléments : il forme le contenu du document.
- des commentaires et instructions de traitement, dont la présence est facultative. Ces derniers pouvant, moyennant certaines restrictions, apparaître aussi bien dans le prologue que dans l'arbre d'éléments.

Un exemple de document XML est présenté en figure 2.1suivante.

```
<? Xml version= «1.0 » encoding = » ISO-8859-1» ? >
<!−−Exemple de fichier XML d´écrivant un article scientifique −−>
<articleannee=”2003”>
<en-tete>
<titre>Recherche d'information sur le web : la grande révolution</titre>
<auteur>André Dupont</auteur>
</en-tete>
<corps>
<section>
<sous-titre>Histoire de l'hypertexte : des pères fondateurs au World Wide Web</sous-titre>
        <par>Afin de maîtriser les enjeux des systèmes hypertexte, il convient,
        même...</par>
```

Figure 2.1- Exemple d'un document XML.

II.3. 2.1 Le Prologue :

Il peut contenir une déclaration XML, des instructions de traitement et une déclaration de type document.

(a). Déclaration XML :

permet donc d'indiquer la version XML utilisée, le jeu de codage de caractères utilisé et l'autonomie du document comme suit :

? xml version : Version XML utilisée dans le document (version 1.0)

- Encoding : Jeu de codage de caractères utilisé ici est : ISO-8859-1 (contient le code ASCII en ajoutant de nombreux caractères accentués occidentaux)
- Standalone : existence ou non de déclarations extérieures au document qui doivent être prises en compte pour le traitement de celui-ci :

  – S'il n'y a pas de DTD ou si elle est interne, le document est autonome et la valeur de l'attribut standalone peut être définie à 'yes'.

  – Si la DTD référencée est externe la valeur de cet attribut doit être définie à 'no'.

  – Si l'attribut standalone est omis, c'est la valeur 'no' qui est prise par défaut.

La déclaration est facultative mais fortement conseillée. Chacune des trois informations est aussi facultative mais si elles apparaissent c'est obligatoirement dans cet ordre.

(b). Instructions de traitement :

Elles sont facultatives et ne font pas partie du document. Leur contenu est simplement transmis à une application en vue de déclencher certains traitements. Une instruction de traitement (figure 2.2) commence par **< ?**et se termine par **?>**, Les instructions de traitement qui servent le plus souvent, sont la déclaration XML ainsi que la déclaration de feuille de style.

```
< ? xml version = «''1.0''» ?>
```

Figure 2.2 – Instruction de traitement XML.

(c). Déclaration de Type de Document (ou **DTD**):

La DTD permet d'établir les règles de définition de la structure d'un document XML, les éléments à inclure, leur type et les valeurs par défaut à leur attribuer. Elle peut être déclarée au sein du document ou sous forme d'un document externe. Bien que facultative, il est souvent intéressant pour un document XML de se conformer à une DTD. Lorsqu'une DTD est associée à un document XML on dit qu'il est valide. Un exemple de DTD est donné en figure 2.3 suivante.

```
< ? xml version="1.0" ?>
< !--DTD pour personne.xml-->
<!ELEMENT personne (nom,  prenom, fonction)>
<!ELEMENT nom (# PCDATA)>
<!ELEMENT prenom (# PCDATA)>
<!ELEMENT fonction (# PCDATA)>
```

Figure 2.3 - déclaration de type DTD.

II.3.2.2 Les Commentaires :

Des commentaires peuvent être insérés dans les documents. Ils sont encadrés par les marques de début **< !--** et de fin **-- !>** de commentaire (exemple en figure 2.4).Le corps du commentaire peut contenir n'importe quel caractère à l'exception de la chaîne <<-->>. On ne peut donc pas inclure un commentaire dans un autre. Le commentaire ne peut pas non plus être inclus à l'intérieur d'une balise.

```
< !-- ceci est un commentaire -- !>
```

Figure 2.4 – un commentaire XML.

II.3.2.3 L'arbre d'éléments :

La partie essentielle d'un document XML sera toujours formée d'une hiérarchie d'éléments qui dénote la sémantique de son contenu: c'est l'arbre d'éléments. Dans un document XML il n'existe qu'un et un seul élément racine, qui contient tous les autres.

Chaque élément d'un document XML se compose d'une balise d'ouverture, d'un contenu d'éléments et d'une balise de fermeture.

Syntaxe :

```
<nom_balise>contenu-élément</nom_balise>
```

Figure 2.5 – un élément XML.

Un élément peut contenir des données, des références à des entités, des sections littérales et des instructions de traitement. Un élément peut avoir un contenu récursif, c'est à dire qu'il peut contenir une instance du même type d'élément que lui-même. Un élément vide est représenté par <nom/>

Remarque: La balise d'ouverture d'un élément peut inclure des attributs sous la forme de paires nom='valeur' (exemple en figure 2.6). La valeur d'un attribut est une chaîne de caractères encadrés par des apostrophes (' ') ou par des guillemets (« »).

```
<employer id = « chf-677 »>AbdEnnour</employer>
```

Figure 2.6 – attribut XML.

II.3.3 Les parseurs **XML:**

Un parseur XML est un analyseur, qui permet d'accéder aux documents XML pour pouvoir en extraire les données et les traiter, ainsi éventuellement de vérifier la validité d'un document. On distingue deux principaux parseurs XML : DOM et SAX.

- **DOM** (Document Object Model) est une spécification du W3C. Représentant le document XML sous forme d'arbre de nœuds typés (de types élément, attribut, ou texte) et liés par des relations de structure (père-fils). Le contenu textuel est situé dans les nœuds feuilles. La figure 2.7 montre un exemple de document XML et sa structure d'arbre DOM.

```
<doc année="2009">
<titre> Approche de RI conceptuelle </titre>
<sect1 >
 <par1> L'indexation conceptuelle se réfère à la construction...< par1>
<par2> Une ressource sémantique est utilisée..... </par2>
</sect1>
<sect2 >Un document est décrit par un arbre de nœuds...
<par3> DOM est une spécification du W3C dont l'objectif.... </par3>
 </sect2>
</doc>
```

Figure 2.7 – Document XML

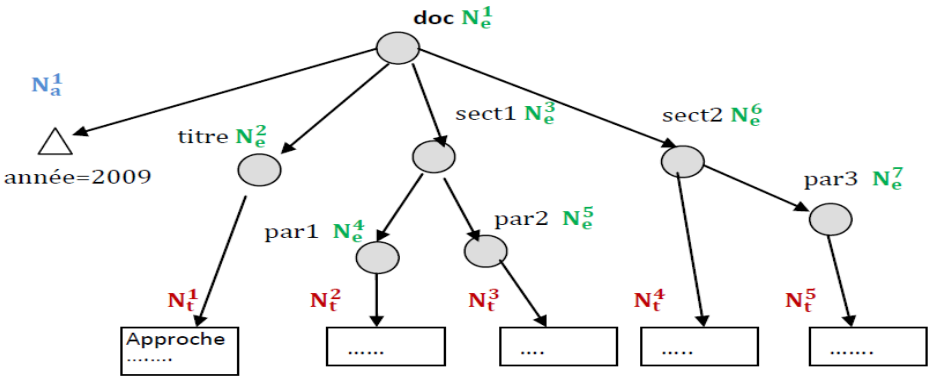


Figure 2.8 – représentation DOM

- **SAX** : est un parseur orienté évènements. Il traite le document XML au fur et à mesure de la rencontre des élèments de documents, et pour chaque élèmnet rencontré un évènement est envoyé (ie. une fonction est appellée). Les évènements envoyés peuvent être des balises ouvrantes, des balises fermantes, des élèments texte, etc…

Remarque :

Le traitement d'un document XML avec DOM est plus facile car l'information est facilement exploitable, mais l'inconvénient est qu'elle occupe de l'espace mémoire surtout si le document est volumineux. Par contre, le parseur SAX utilise moins de mémoire car il n'accumule aucune donnée dans une structure, mais il nécessite des traitements

supplémentaires pour traiter un document XML (par exemple : sauvegarder des données au moment de parcoure d'un nœud dans un document pour les traiter plus tard).

II.4 Problématiques de la Recherche d'Information Semi-Structurée

La Recherche d'Information semi-structurée (RIS) est différente de la RI Classique en plusieurs aspects :

- L'organisation de l'information en unités (éléments) plus fines qu'un document, réactualise la granularité de l'information à retourner à l'utilisateur, et c'est plus judicieux de retourner seulement ces éléments, voir un ensemble d'éléments issus de documents différents, et cela pour éviter que l'information utile ne soit trop dispersée.
- L'expression des besoins utilisateur qui peut se porter sur le contenu ou/et la structure.

La figure suivante (tableau. 2.1) résume les principaux critères de comparaison entre la RI classique et la RI semi-structurée.

	RI Classique	RI Semi-Structurée
Type de document	Document non structure : document plat	Document semi-structuré
Contenu	Texte seulement	Texte + structure
Unité de recherche	Document	Elément
Type de requête	Contenu	Contenu et/ structure
Langage de requête	Langage libre	Langage structuré

Tableau 2.1 – comparaison entre la RI Classique et RI Semi-structurée.

Pour prendre en considération ces différents aspects, les techniques de la RI Classique doivent être adaptées et pourquoi pas proposer de nouvelles techniques de recherche d'information semi-structurée qui puissent répondre aux problématiques particulières suivantes:

1. La première problématique concerne l'indexation : la structure du document entrant en jeu dans le processus d'indexation, les questions suivantes se posent [sauvagnat, 04]:
   * Comment indexer la structure des documents ?
   * Que doit-on indexer de la structure des documents ?
   * Comment relier cette structure au contenu du document ?
   * Quels sont les paramètres à considérer pour la pondération des termes d'indexation ?
   * Quelles structures d'index utiliser ?

2. La seconde problématique concerne les langages d'interrogations qui doivent permettre d'utiliser des requêtes orientées contenu et/ou structure et d'une manière simple. Avec l'absence de la structure dans les requêtes, les systèmes doivent décider de la granularité idéale de l'information à renvoyer à l'utilisateur, les conditions de la structure donnent des indications sur le type d'élément à renvoyer.

3. La dernière problématique concerne les modèles de recherche et de tri des unités d'information. Cette problématique est liée au score de pertinence attribué à l'unité d'information (un nœud de document XML) vis-à-vis d'une requête, la pertinence est évaluée selon les deux notions : l'exhaustivité (l'élément doit contenir toutes les informations requises par la requête) et la spécificité (le contenu de l'élément concerne la requête), en plus l'élément renvoyé se doit d'être auto-explicatif (l'élément peut être compris indépendamment du reste du document). Cependant en prenant compte de ces trois notions (l'exhaustivité, la spécifié et l'auto- explicativité) l'élément à renvoyer doit être *la plus petite unité d'information cohérente pertinente pour la requete*[Piwowarski, 03].

II.5- Indexation des documents semi-structurés

Dans ce qui suit, nous proposons de décrire les approches d'indexation des documents XML, ainsi que les structures d'index utilisées.

II.5.1-Approches d'indexation des documents XML

Les approches d'indexation des documents sont classifiées en approches d'indexation basées sur le contenu et approches d'indexation basées sur la structure.

II.5.1.1 Approches basées sur le contenu ou indexation de l'information textuelle :

L'unité d'information pertinente dans les documents semi-structurés correspond à l'élément XML. Ainsi, l'indexation du contenu textuel d'un document semi-structuré consiste à indexer les éléments du document. Cependant, si beaucoup considèrent les éléments comme des unités disjointes sans connexion entre elles, d'autres propagent certains termes ou tous les termes d'un élément donné vers ses ancêtres (technique dite des sous arbres imbriqués). On distingue ainsi deux classes d'approches d'indexation du contenu :

1. **Les approche par sous arbres imbriqués** :

Elles indexent le contenu textuel en attribuant un poids, puis en propageant chaque terme de l'index vers ses ancêtres en diminuant son poids selon la distance entre le nœud qui le

contient et celui vers lequel le terme est propagé. L'index contient des informations redondantes car les nœuds de l'index sont imbriqués les uns dans les autres.

2. **Les approches par unités disjointes** :

Elles décomposent le document en unités disjointes, c'est-à-dire elles n'indexent que les éléments contenant du texte sans propager le contenu aux ancêtres.

II.5.1.2 Approches basées sur la structure ou indexation structurelle :

L'indexation de la structure passe par le choix du mode de représentation des éléments structurant les documents. L'information structurelle peut être indexée selon des granularités variées, c'est-à-dire que toute l'information structurelle n'est pas forcement utilisée dans le processus d'indexation. On trouve :

1. **Indexation basé sur les champs** :

On restreint la structure de l'arbre au nœud feuille qui la compose en associant à chaque balise les termes indexés qu'elle contient avec leur fréquence, l'inconvénient est qu'elle ne sauvegarde pas les relations hiérarchique entre les nœuds.

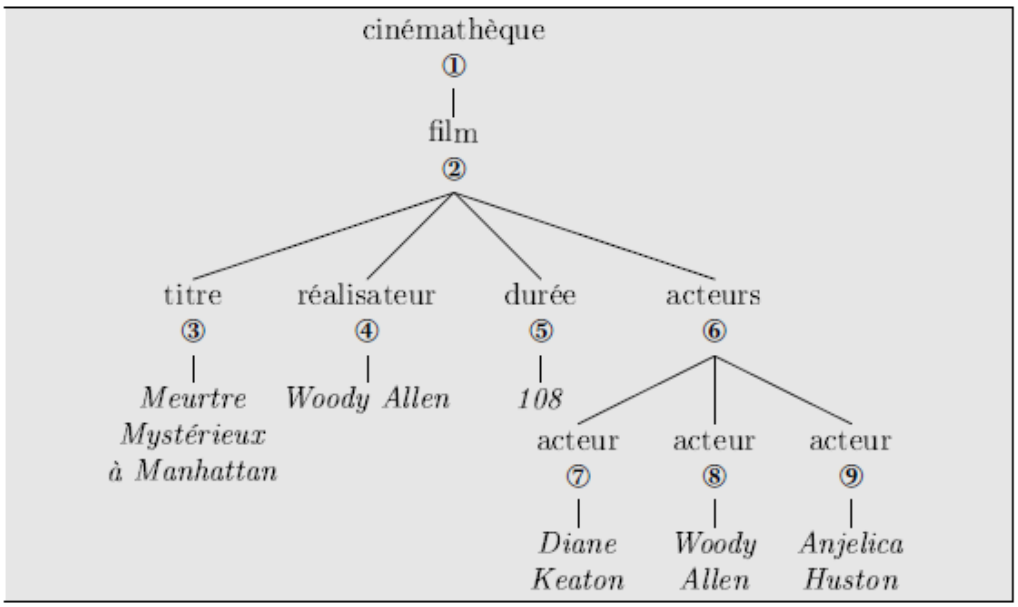


Figure 2.9 – exemple de document représenter sous forme d'arbre XML

Terme	Sans propagation	Avec propagation
Meurtre	Titre	cinémathéque, film, titre
Allen,	réalisateur, acteur	cinémathèque, réalisateur, film, titre, acteurs, acteur

Tableau 2.2 – exemple d'indexation basé sur les champs de la figure 2.9.

2. L'indexation basée sur les chemins :

Les noms de balises sont remplacés par les chemins d'accès où leur notation est basé sur XPath, l'avantage de cette méthode est d'accélérer le processus de recherche pour une information se situant dans plusieurs balises qui portent le même nom .L'inconvénient est qu'elle ne représente pas les relations de descendance des différents éléments du document XML.

Terme	Sans propagation	Avec propagation
Meurtre	(/cinémathèque /film /titre)	(/cinémathèque) (/cinémathèque /film) (/cinémathèque/film /titre)
Allen	(/cinémathèque/film/réalisateur)	(/cinémathèque) (/cinémathèque/film) (/cinémathèque/film/réalisateur) (/cinémathèque/film /acteurs /acteur) (/cinémathèque/film /acteurs)

Tableau 2.3 – indexation basés sur les chemins de la figure 2.9.

3. indexation basés sur les Arbres :

La spécificité de cette méthode est d'attribuer un identifiant à chaque nœud d'arbre, pour permettre de le localiser d'une façon précise et trouver les relations hiérarchiques entre les éléments, L'identifiant unique peut également être, tout simplement, le chemin d'accès (XPath absolu, avec le numéro des éléments) de l'élément.

Terme	Sans propagation	Avec propagation
Meurtre	(/cinémathèque [1]/film [1]/titre [1]) (3)	(/cinémathèque [1]) (1) (/cinémathèque [1]/film [1]) (2) (/cinémathèque [1]/film [1]/titre [1]) (3)
Allen	(/cinémathèque [1]/film [1]/réalisateur [1]) (4)	(/cinémathèque [1]) (1) (/cinémathèque [1]/film [1]) (2) (/cinémathèque [1]/film [1] /réalisateur [1]) (4) (/cinémathèque [1]/film [1]/acteurs [1]/acteur [2]) (7) (/cinémathèque [1]/film [1]/acteurs [1]) (8)

Tableau 2.4- indexation basés sur les arbres de la figure 2.9.

Nous présentons si dessous quelques méthodes d'identification :

1. Codage Préordre / postordre :

Le codage pré/post ordre est une paire (pré, post) attribuée à un nœud, où 'pré' est la valeur du préordre et 'post' est la valeur du postordre du nœud.

- La valeur préordre (parcours préfixe) : consiste à parcourir l'arbre de haut en bas, de gauche à droite et d'attribuer un numéro à chaque nœud rencontré et ensuite l'incrémenter.

- La valeur postordre (parcours postfixe): est de parcourir l'arbre de bas en haut, de gauche à droite et d'attribuer un numéro à chaque nœud rencontré puis l'incrémenter.

Exemple :

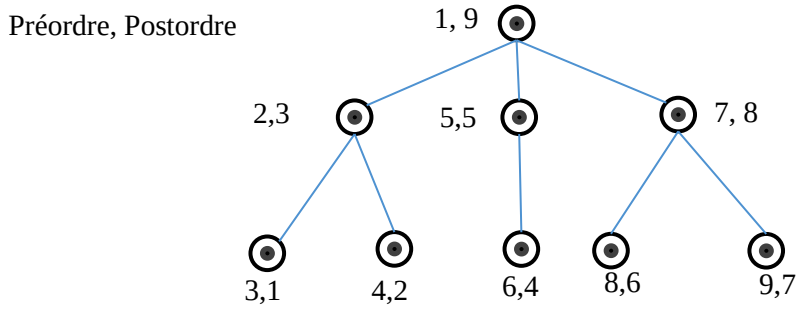


Figure 2.10 : numérotation post/pré ordre

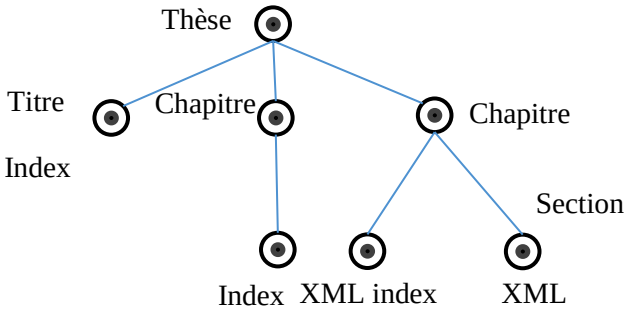
Avec cette annotation on peut identifier les rapports ancêtres descendants avec les relations suivantes :

n_1 est ancêtre de n_2 : pré (n_1) <pré (n_2) et post (n_1) > post (n_2).

L'inconvénient de cette méthode et de recalculer tous les identifiants des nœuds pour chaque nouvelle insertion d'un nouveau nœud.

2. Nombre de frères (*Siblingnumbers*)

Pour identifier un nœud de document sans équivoquement, le chemin de la racine menant à ce nœud est représenté par [Sacks,94] comme suit: Pour chaque nœud sur le chemin, le nombre de ses frères (fils du même parent) précédents du même type (avec la même étiquette) est calculé. C'est son nombre de '*Siblings'*. Les étiquettes entrantes de tous les nœuds sur le chemin de la racine forment ensemble un chemin d'étiquette. Chaque étiquette est annotée avec le nombre de siblings du nœud qu'il mène. Le chemin étiquette/sibling résultant est une sorte de chemin de position. La Figure 4 montre une petite.



index	/Thèse/titre ; /Thèse/chapitre(0)/section(0) ; /Thèse/chapitre(0)/section(1);
XML	/Thèse/chapitre(0)/section(1) ; /Thèse/chapitre(1)/section;

Figure 2.11

Comme on peut le voir dans La Figure 2.11, les enfants du même nœud avec des étiquettes identiques ne peuvent pas être confondus. Par exemple, les deux nœuds chapitre directement au-dessous de la racine du document sont mentionnés comme /Thèse/chapitre [0] et /Thèse/chapitre [1], respectivement.

II.5.2-Structures d'Index

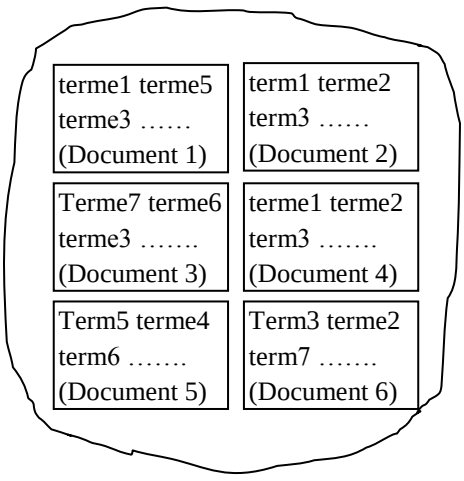
La qualité et l'efficacité de la réponse retournée par le SRI à l'utilisateur dépend de la manière dont l'index doit être organisé (stocké). Seules les structures d'index peuvent déterminer la méthode d'indexation et agissent directement sur les performances du SRI. Nous présentons ici quelques structures d'organisation de l'index :

II.5.2.1 Les fichiers séquentiels :

Un fichier séquentiel stocke les fichiers de données les uns aux autres selon l'ordre de leur insertion, c'est le moyen le plus simple pour stocker un fichier de données, les enregistrements logiques sont accessibles séquentiellement, autrement dit pour accéder à un enregistrement de rang N, il faut parcourir les N-1 enregistrements précédents ce qui n'y pas performent.

II.5.2.2 Les fichiers inverses :

La structure des fichiers inverses est la base de tous les SRI. Le fichier inverse permet d'accéder aux documents contenant les mots clés d'une requête sans avoir à parcourir l'ensemble des documents. Comme l'index d'un ouvrage, les entrées de l'index sont les mots du corpus, qui pointent vers la liste des documents qui les contiennent. On conserve généralement le nombre de documents qui contiennent ce mot (df) ou d'autres informations comme le nombre d'occurrences du mot dans chaque fichier et la position de ces occurrences. La figue 2.12 est une représentation d'un fichier inverse.



Collection des documents

terme1 : document1, document2, document4.

terme2 : document2, document4, document6.

…

Fichier inverse de la collection des documents

Figure 2.12 – la représentation d'un fichier inverse

Ce type de fichier, même s'il est rapide pour trouvère un résultat, consomme énormément d'espace mémoire et les mises à jour sont aussi couteuses puisqu'il faut refaire l'index à chaque ajout.

II.5.2.3 Les fichiers de signature :

Les insuffisances des fichiers inverses ont suscité à chercher des solutions. Cela amène à revoir les fichiers séquentiels pour les adapter à une représentation efficace dans tous les domaines.

Le fichier de signature est une chaine de bit qui représente les mots clés de la collection, où chaque bit est réservé pour chaque mot clé. L'index d'un document de la collection est une suite de chaine de bit, où les bits des mots clés appartenant à ce document sont mis à '1'. La position d'un mot clé dans cette chaine est calculée grâce à une fonction de hachage.

La réponse à une requête consiste à calculer sa chaine de bits et la comparer avec celle calculée pour chaque document.

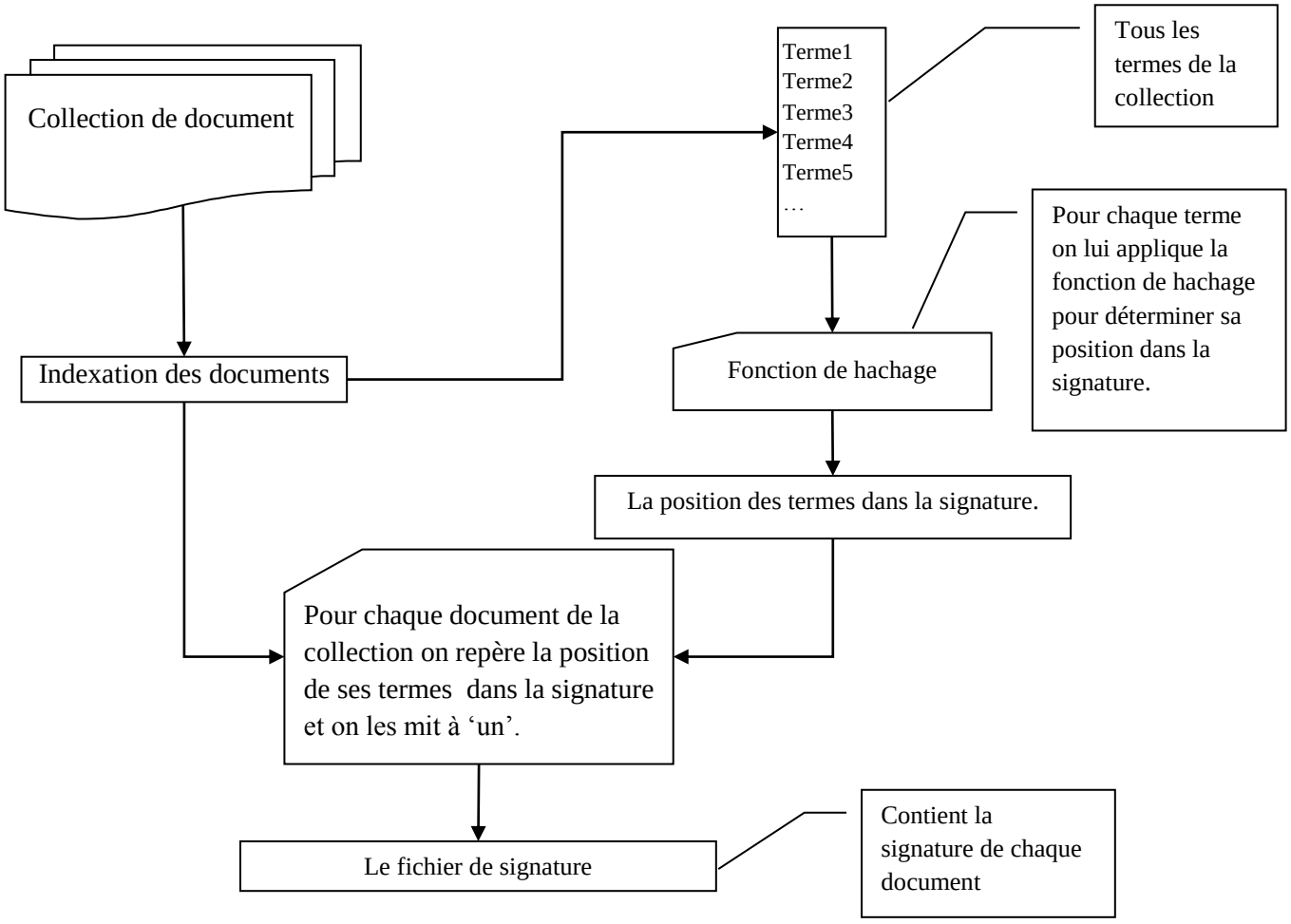


Figure 2.13 – processus de création de fichier de signature

II.5.3 Pondération des termes d'indexation :

Dans le cadre des documents XML, l'importance du terme au niveau de l'élément vient s'ajouter à celui de son importance au niveau local au sein du document et celui globale au sein de la collection. Les formules tf*idf sont adaptées pour prendre en considération la force discriminatoire d'un terme *t* pour une balise *b* relative à un document *d* qui est présenté sous forme tf.itdf(TermFrequency-Inverse Tag and Document Frequency), les paramètres pris en compte pour calculer l'importance du terme au niveau de l'élément sont:

- Le nombre de nœuds qui contiennent le terme.
- Le nombre de nœuds dans la collection.
- La longueur du nœud qui contient le terme.
- La longueur moyenne des nœuds feuilles dans la collection.
- La fréquence du terme dans le nœud.

Quelques formules de pondération :

$$w_{T,E}^{t} = tf_{T,E} * idf_{T}^{t} = \frac{freq_{T}(E)}{\max(freq(E))} * (\log\left(\frac{|E^{T}|}{n_{T}}\right) + 1)$$

Formule 2.1

Avec

- freqT (E) le nombre d'occurrences de T dans E.
- maxfreq(E) le nombre maximal d'éléments de la collection possédant la même étiquette que E.
- |Et| le nombre d'éléments de type t et nT le nombre d'éléments contenant T. [Schileder,02]

Azevedo et al. [2004]

$$w_{D}(t_i, e) = log(tf(t_i, e)) * log(N * idf(t_i, e))$$

Formule 2.2

Où

- tf (ti; e) calcule le nombre d'occurrences du terme ti dans e.
- idf(ti; e) est l'inverse du nombre d'éléments contenant ti.
- N est le nombre total d'éléments dans la collection.

Hatano& al [Hatano& al, 2002]

$$w(t,e) = \frac{tf_t(e)}{\sum_{t' \in T} \sum_{e' \in E} tf_{t'}(e')} * log\frac{N}{n_t}$$

Formule 2.3.

w(t,e) : poids du terme *t* dans l'élément *e.*
tft(e) : fréquence du terme *t* dans l'élément *e.*
T : ensemble du vocabulaire.
E : ensemble des éléments du document considéré.
N : Nombre total de documents dans le corpus (collection).
nt: Nombre total de documents contenant le terme *t*.

Sauvagnat [Sauvagnat, 2005],

w_{inf}= poids du terme *i* dans le nœud feuille n_f est calculé par la formule suivante

$$\mathbf{W_i = tf_i^{nf} * Ief_i}$$

Formule 2.4

Ou tf_i^{nf} c'est la fréquence du terme *i* dans le nœud feuille *nf* et *iefi* Inverse Element Frequency se calcule comme suit :

$$\mathbf{ief_i = \log\left(\frac{N}{n+1}\right) + 1}$$

Formule 2.5

Où

- N mesure le nombre total de nœuds feuilles
- n est le nombre des nœuds feuillent contenant le terme i.

Une autre formule pour calculer Le poids d'un terme dans les nœuds feuilles est calculée comme suit :

$$Wi,j = tf_{ij} * ief_j = \frac{nbocc(t_i,f_j)}{nbterm(f_i)} * \log\left(\frac{|F_c|}{|nf_j|}\right) / \log(|F_c|).$$

Formule 2.6

- $nbocc(t_i,f_j)$ nombre d'occurrence de terme t_i dans le nœud feuille f_j.
- $nbterm(f_i)$nombre de terme dans le nœud feuille.
- $|F_c|$ nombre de nœud feuille de la collection.
- nf_j nombre de nœud feuille contenant le terme t_i.

Sauvagnant 2006

$$W_{i,j} = tf_{ij} * ief_j = nbocc(t_i, f_j) * \log\left(\frac{|F_c|}{|nf_i|}\right)$$

Formule 2.8

- $nbocc(t_i, f_j)$ nombre d'occurrence de terme t_i dans le nœud feuille f_j.
- $|F_c|$ nombre de nœuds feuilles de la collection.
- nf_j nombre de nœuds feuilles contenant le terme t_i.

II.6 Interrogation

Le processus d'interrogation est composé de trois étapes :

- La formulation du besoin par l'utilisateur peut être effectuée de plusieurs manières (que nous citerons plus bas), puis cette requête sera représentée avec une structure semblable à celle des documents pour pouvoir les apparier.
- L'appariement requête/élément : consiste à attribuer des scores de pertinence à chaque élément du document par apport à la requête grâce à une fonction d'appariement.
- Le tri par ordre de pertinence et la présentation des résultats.

Les requêtes utilisateur peuvent être exprimées de deux manières :

- Requêtes orientées contenu : consistent à introduire de simples mots clés où le système fera une recherche complète sur la collection en parcourant tous les éléments.
- Des requêtes orientées contenu et structure : où l'utilisateur peut poser des conditions sur la structure d'une manière précise, ou bien d'une manière vague et cela en ayant déjà des connaissances préalables sur la structure des documents.

II.7 Les modèles de recherche :

Dans cette section, nous présentons le modèle vectoriel étendu adapté aux documents semi-structurés.

Les mesures de distance dans un espace vectoriel sont utilisées pour calculer la similarité entre l'élément et la requête qui sont représentés Par des vecteurs de termes pondérés.

En générale les approches du modèle vectoriel propagent les termes des nœuds feuilles dans l'arbre du document.

La similarité d'un nœud ***n*** à une requête **q = {t1, t2, ...tT}** est exprimée selon la formule suivante :

$$sim(q,n) = \alpha(T)cosm(q,n) + \sum_{k=1}^{s} \frac{cosm(q,n_k)}{\beta^{k-1}}$$

Formule 2.9 : fonction de similarité du modèle vectoriel étendu

Où

α(T) :est un facteur permettant de prendre en compte le type du nœud,
S :est le nombre de nœuds enfants n_k de ***n***,
B :est un paramètre permettant d'assurer que le nombre d'enfants n'introduit pas un biais dans la formule.

La fonction ***cosm*** est définie de la façon suivante :

$$Cosm(q,n) = \sum_{i=1}^{T} \frac{w_i^q * w_i^n}{|n|}$$

Formule 2.10

Où :

- w_i^qet w_i^q respectivementle poids du terme t_i dans la requête ***q*** et dans le nœud ***n***,
- |***n***| est le nombre de termes dans le nœud ***n***

(Hlaoua et al., 2006), (Sauvagnat, 2005) :

Le modèle est basé sur une méthode de propagation de pertinence. Des valeurs de pertinence sont d'abord calculées pour les différents nœuds feuilles (c'est à dire les nœuds contenant du texte). Ces valeurs sont par la suite propagées et agrégées vers les nœuds ancêtres en prenant compte de la distance séparant les nœuds ancêtre des nœuds descendant. La pertinence d'un nœud nfpour une requête composée de $m$termes q= {$t1$, $t2$,…, tm} est exprimée selon l'équation suivante :

$$RSV(Q,nf) = \sum_{j=1}^{j=m} w_j^Q * w_j^{nf}$$

Formule 2.12

Où

- $w_j^Q = tf_j^Q$*iefi est le poids du terme $t_j$dans Q

- $w_j^{nf} = tf_j^{nf} * idf_j * ief_j$ est le poids du terme tjdans nf.

$$\text{Avec} \quad \text{ief}_i = \log\left(\frac{N}{n}\right)$$

Formule 2.13

Où

- N et le nombre total des nœuds feuilles
- n le nombre des nœuds feuille contenant le terme i.

$$\text{Avec idf} = \log\left(\frac{|D|}{|di|}\right)$$

Formule 2.14

Où

- |D| est le nombre total de document de la collection
- $|d_i|$ est le nombre de documents contenant le terme ti

La valeur de pertinence d'un nœud interne *n* est calculée comme suit :

$$RSV(Q,n) = \left|F_n^p\right| * \sum_{k=1..F_n} \alpha^{dist(n,nf_k)-1} * RSV(q,nf)$$

Formule 2.15

Où

- nf_k sont les nœuds feuilles descendants du nœud *n*,
- dist(n, nf_k) est la distance entre le nœud n et le nœud feuille nf_k dans l'arbre ,
- $|F_n^p|$est le nombre de nœuds feuilles descendants du nœud n ayant un score différent de zéro,
- F_n est le nombre total de nœuds feuilles descendants de n,
- $\alpha \in [0,1]$ est un paramètre qui quantifie l'importance de la structure dans l'évaluation du score de pertinence.

II.8 Evaluation : la compagne d'évaluation INEX

II.8.1 INEX :

INEX *(Initiative for the Evalution of the XML Retrieval)*est actuellement la seule compagnie d'évaluation des différents systèmes de recherche d'information pour des documents XML .Le but principale d'INEX est de promouvoir l'évaluation de la recherche sur les documents XML en fournissant des collections de test (par exemple, une collection d'articles IEEE en 2002 et 2005, des documents Wikipédia en 2006-2010, et une collection de livres numérisés sous licence de Microsoft à partir de 2007) des mesures d'évaluation uniformes, et un forum pour tous les organisations qui y participent à comparer leurs résultats et à améliorer leurs stratégies. La collection de test est constituée d'un ensemble de document

XML, un ensemble de besoin en information (requête) et des repenses à ces besoins informationnels.

II.8.2 La structure du document INEX :

A partir de 2009, INEX utilise une nouvelle collection de documents basée sur Wikipedia. La collection INEX 2009 est d'environ 50,7 Go en taille repartie sur 995 parties. Cette collecte est de 8,6 fois la taille de la collecte prévue en 2007 et en 2008.

Cette collection contient un certain degré de structure uniforme comme celle des documents XML,mais ne suivant pas strictement la DTD (Document Type Définition) d'où la collection est considérée comme semi-structurée .La collection actuelle a plus de 30.000 balises dont la plus part sont éliminées lors de l'analyse. La structure d'un document INEX 2009 est comme suit :

```
<? Xml version = "1.0" encoding = "UTF-8"?>
<! - Généré par CLiX / Wiki2XML [MPI-INF, MMCI @ UdS] $ LastChangedRevision: 92
$ Sur 17.04.2009 3:27:08 [mciao0828] ->
<! DOCTYPE article SYSTEM "../ article.dtd">
<Article xmlns: xlink = "http://www.w3.org/1999/xlink">
<Header>
<Title> Portail: football anglais image / Sélectionné / 21 </ title>
<Id> 16183995 </ id>
<Révision>
<Id> 196856544 </ id>
<timestamp> 2008-03-08T21: 32: 02Z </ timestamp>
<Contributor>
<username>Jardin</ username>
<Id> 5019622 </ id>
</ Contributor>
</ Révision>
<Catégories>Portail de football<category> Anglais photos sélectionnées </ category>
</ Catégories>
</ Header>
<bdy>
```

Figure 4.1: exemple d'un document de la collection de INEX 2009.

II.8.3 La structure de la requête (topic) INEX :

```
<topic id="2009001" ct_no="186">
<title>Nobel prize</title>
<castitle>//article[about(., Nobel prize)]</castitle>
<phrasetitle>"Nobel prize"</phrasetitle>
<description>information about Nobel prize</description>
<narrative>
        I need to prepare a presentation about the Nobel prize. Therefore, I want to collect
        information about it as much as possible. Information, the history of the Nobel prize or the
        stories of the award-winners for example, is in demand.
</narrative>
</topic>
```

Figure 4.2 – exemple d'une requête INEX 2009

Les requêtes INEX se divisent en deux catégories principales :

- Les CO (content only) sont des requêtes courtes composées de simples mots clés.
- Les CAS (content and structure) ces requêtes contiennent des contraintes sur la structure des documents

Chaque requête est constituée d'un ensemble de champs qui explique le besoin informationnel de l'utilisateur

- Le champ <Castitle> : est une courte explication du besoin informationnel en précisant tout changement structurel. Aussi connu comme le contenu de la structure (CAS), Il donne la forme structurée de la requête ;

- Le champ <Title> : donne une courte explication du besoin d'information en utilisant des mots clés simples. Il sert comme résumé au contenu informationnel.

- Champs <Description><Narrative> : donne une explication détaillée de la nécessité de l'information et de ce qui fait un élément pertinent. Le <Narrative> ne devrait pas seulement expliquer l'information recherchée mais aussi le contexte et la motivation de la nécessité de l'information, il est important que cette description soit claire est précise. La présence de ces deux champs est optionnelle.

II.8.4 La tache ad-hoc [sauvagnat, 06]:

La tâche principale d'INEX est la tâche de recherche *ad hoc*. Comme en recherche d'information traditionnelle, la recherche ad hoc est considérée dans INEX comme une

simulation de l'utilisation d'une bibliothèque, où un ensemble statique de documents est interrogé avec des besoins utilisateurs, c'est-à-dire des requêtes. Les requêtes peuvent contenir à la fois des conditions structurelles ou de contenu, et en réponse à une requête, des éléments (et non forcément des documents) peuvent être retrouvés à partir de la bibliothèque. La tâche *ad hoc* se divise en 2006 en quatre sous-tâches :

– **la tâche de recherche exhaustive** (*thoroughtask*) : qui consiste à retourner tous les éléments (éventuellement imbriqués les uns dans les autres) répondant au sujet de la requête et triés par degré de pertinence.
– **la tâche de recherche focalisée** (*focusedtask*), qui consiste à retourner les éléments les plus précis possibles satisfaisant le besoin en information de l'utilisateur (éléments imbriqués interdits).

– **la tâche de recherche de pertinence en contexte** (*relevant in contexttask*) : qui consiste à retourner des éléments dans le contexte d'articles entiers (les unités d'informations sont triées par document).

– **la tâche de recherche du meilleur en contexte** (*best in contexttask*) : qui consiste à renvoyer un seul élément par article, à savoir le meilleur point d'entrée dans l'article. La formule utilisée est la même que la précédente.

Nous utilisons la collecte, les requêtes fournies par INEX et notre logiciel pour produire les éléments XML pertinents vis-à-vis de notre système, les résultats retournés sont évalués en utilisant les mesures d'évaluation de pertinence fournis par INEX.

II.8.5 Les jugements de pertinence :

Les jugements concernant la pertinence des éléments renvoyés par les systèmes sont effectués par les participants eux-mêmes à l'aide d'une interface mise à disposition en ligne, (Lalmas et al.,2005), (Piwowarski et al.,2008).

Pour faire les jugements de pertinence, les dimension d'exhaustivité et de spécificité ont été utilisées (Malik et al., 2005) . L'exhaustivité est jugée par l'utilisateur sur une échelle à quatre niveaux (Lalmas et al., 2005) :

- Non exhaustive : aucune correspondance entre la requête et l'élément ;
- Partiellement exhaustive : l'élément ne traite que certains aspects de la requête ;
- Très exhaustive : traite tous ou presque tous les aspects de la requête ;
- Trop petite : l'élément peut être considéré comme pertinent àcertains égards, mais la partie pertinente est trop petite pour être considérée.

Ainsi pour la spécificité une échelle à quatre niveaux est proposée :

- Pas spécifique : le sujet de la requête n'est pas un thème de l'élément ;
- Marginalement spécifique : le sujet de la requête est un thème mineur de l'élément;
- Assez spécifique : le sujet de la requête est un thème majeur de l'élément ;
- Très spécifique : le sujet de la requête est le thème de l'élément.

II.8.6 Mesures d'évaluation :

Les mesures traditionnelles utilisées dans la RI classique ont été adaptées pour prendre en considération le besoin supplémentaire induit par la recherche d'information semi-structurée, nous présentons dans ce qui suit les mesures utilisées dans INEX 2009:[NAFFAKHI et al, 13]

– **Précision interpolée selon quatre niveaux de rappel sélectionnés:**$_iP[_jR]$,j∈[0, 00; 0, 01; 0, 05; 0, 1] La précision à un rang *r* est définie Comme suit :

$$P[r]\frac{\sum_{t=1}^{r} rsize(p_i)}{\sum_{t=1}^{r} size(p_i)}$$

Formule 2.16

Avec :

1- p_i est la partie du document assignée au rang i (avec i <= r) dans la liste de résultats L_q des parties de documents retournées par un système de recherche pour une requête q.
2- $rsize(p_r)$ est la taille du texte pertinent contenu dans p_r en nombre de caractères (ce texte est détermine grâce aux jugements de pertinence qui contiennent le bon élément avec sa taille) et size(p_r) est la taille totale du texte contenu dans p_r en nombre de caractères.

Le rappel à un rang r est défini comme suit :

$$R[r]\frac{\sum_{t=1}^{r} rsize(p_i)}{Trel(q)}$$

Formule 2.17

Où *Trel*(*q*) est la quantité totale du texte pertinent pour une requête *q*.

– Moyenne des précisions moyennes interpolées selon 101 niveaux de rappel (***MAiP***), Elle est utilisée pour la mesure du degré de pertinence dans une recherche ciblée [***LAITANG et al, 13***], Pour n requêtes, ***MAiP*** est calculer comme suit :

$$MAiP = \frac{1}{n}\sum_{t} AiP\,(t)$$

Formule 2.18

Où

- ***Aip*** est la précision moyenne interpolée, elle est obtenue par la moyenne des scores de précision interpolée selon 101 niveaux standards de rappel :

$$\boldsymbol{AiP} = \frac{1}{100}\sum_{x=0.0;0.01;\ldots;1.0} iP[x]$$

Formule 2.19

Où

iP[x] est la précision interpolée pour le rappel ***x***

la Précision interpolée à 1% de rappel, aussi connu comme ***iP[0,01]***, est la métrique utilisée pour l'évaluation de la tache ciblée.

Recall : est le terme utilisé pour indiquer la mise en évidences du texte récupéré.
La précision : est le terme utilisé pour indiquer la fraction des textes récupérés.

La formule pour trouver la précision interpolée$\boldsymbol{iP}[\boldsymbol{x}]$:

$$iP[x] = \begin{cases} \max\limits_{l \le r \le |L_q|} (P[r] \cap R[r]) \;\; if\; x \le R[|L_q|] \\ 0\; sinon \end{cases}$$

Formule 2.20

Ou

- $\boldsymbol{L_q}$: la liste ordonnée d'éléments
- $|\boldsymbol{L_q}|$: la taille de la liste ordonnée d'éléments
- ***P[r]*** : précision au rang r
- ***R[r]*** : le rappel au rang r
- $\boldsymbol{R[|L_q|]}$: rappel sur tous les documents récupérés

Pour mesurer l'exhaustivité des résultats obtenus, les formules suivantes sont utilisées :

Précision généralisé: GP [r]: somme des scores de documents jusqu'au document du rang ***r***, divisé par le rang r.

$$gP[r]\frac{\sum_{t=1}^{r} S(d_i)}{r}$$

Formule 2.21

Où

- ***S (d)*** : est le score de document individuel.

Rappel généralisé: GR [r]: Nombre de documents pertinents récupérés jusqu'au document du rang ***r***, divisé par le nombre total de documents pertinents

$$gR[r] = \frac{\sum_{t=1}^{r} IsRel(d_i)}{Nrel}$$

Formule 2.22

Où

- **Nrel** : est le nombre des documents pertinents pour une requête donnée ***q***,
- **IsR(*di*) :** est un réel qui vaut **1** si le document ***d*** de position ***i*** est pertinent pour la requête ***q***, sinon **0**.
- ***AgP***(*AverageGeneralizedPrecision*): La moyenne des précisions généralisée ***AgP*** mesure la performance globale.Cette mesure est calculée comme suit :

$$\boldsymbol{AgR[r]} = \frac{\sum^{|L|}_{r=1} IsRel(d_r).gP[r]}{Nrel}$$ Formule 2.23

La précision généralisée moyenne arithmétique (***MAgP***) est simplement la moyenne de la moyenne des scores de précision généralisée sur tous les sujets, elle permet de mesurer l'exhaustivité des résultats obtenus. La MAgP est une généralisation de la précision et du rappel. Cette mesure est utilisée dans le cadre d'une recherche pertinente dans le contexte.

En supposant qu'il y'a n sujets :

$$\boldsymbol{MAgP} = (1/2) * \sum_{1..n} AgP(t)$$

Formule 2.24

II.9 Conclusion

Dans ce second chapitre nous avons présenté les modèles et algorithmes fondamentaux utilisés en recherche d'information structurée.

L'information est considérée avec une autre gradualité que le document tout entier, cela vu la dimension structurelle apportée au contenu textuel.

L'adaptation des modèles de la RI traditionnelle à la RI classique voit naitre quelques problèmes tels que : (la pondération des termes, l'attribution des scores de pertinences aux nœuds des documents XML, le traitement des conditions de structure…).

Dans le chapitre suivant nous présenterons une approche qui essayera de résoudre ces problèmes ou du moins les minimiser.

Chapitre 3
L'Approche Implémentée : un DataGuide annoté avec un fichier inverse étendu

III.1 Introduction

L'indexation est l'une des phases les plus importantes d'un SRI, sa performance et son exactitude de renvoyer les résultats sont basées sur la manière dont l'indexation est faite. Cette maniéré d'indexer fait référence à comment les descripteurs sont extraits et comment ils sont structurés et sauvegardés. Pour une bonne indexation on doit respecter certains critères, les descripteurs doivent représenter d'une façon fidèle le contenu sémantique des documents et l'organisation des index doit répondre à ces critères suivants:

- Rapide dans l'évaluation de la requête utilisateur.
- Rapide pour la mise à jour de l'index ;
- Permettre la manipulation de gros fichiers de données ;
- Facile à mettre en œuvre ;
- Occuper moins de place que les fichiers de données.

Dans ce chapitre nous présentons une approche d'indexation qui est le « DataGuide » proposée par Fouad Dahak, à laquelle nous avons apportée quelques modifications. Que nous citons plus bas dans ce chapitre.

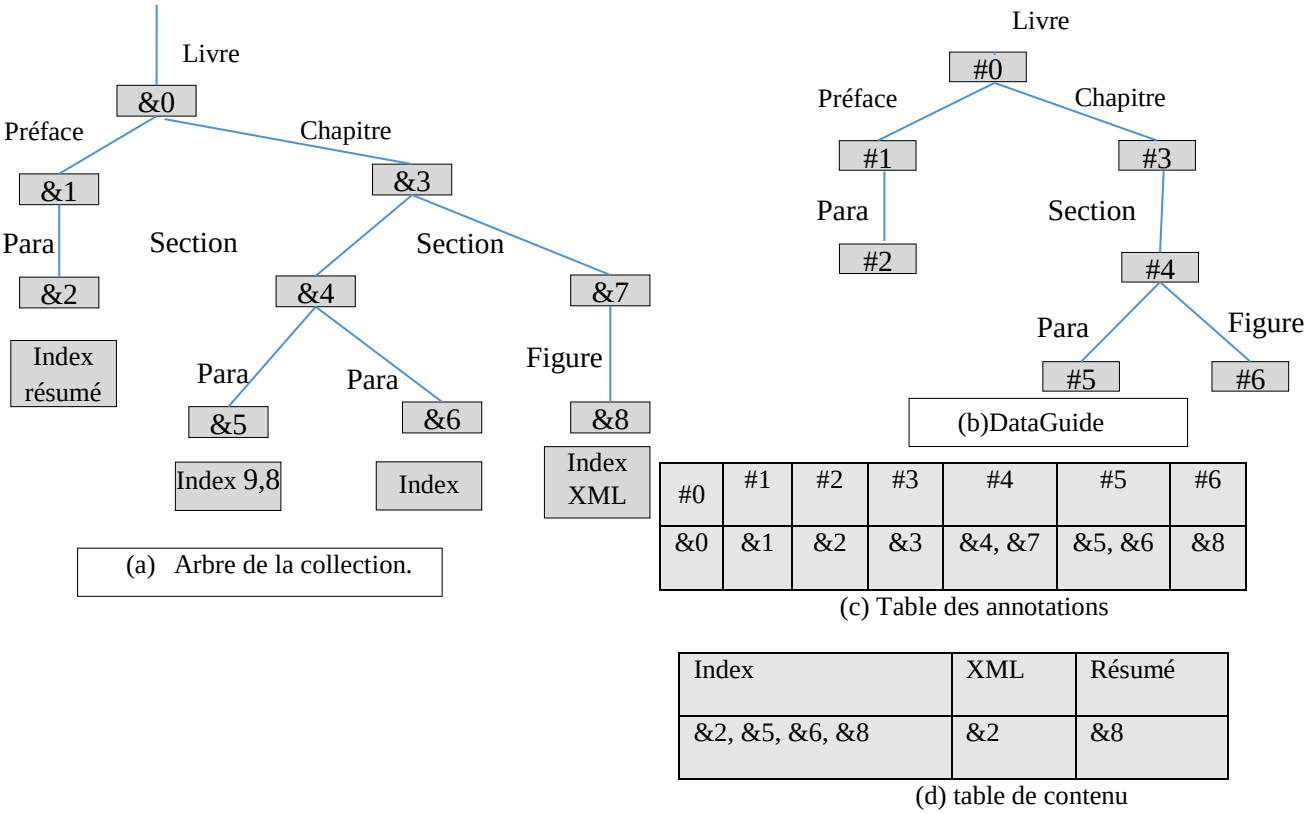
III.2 Le DataGuide

Le *DataGuide* ([Goldman, 97]) est une structure d'index compacte pour la représentation de documents semi-structurés sous forme d'arbre ou de graphe basé sur l'arbre de la collection des documents[1].

Pour une collection du document, le DataGuide est un graphe tel que chaque chemin de l'arbre de la collection est représenté exactement une seule fois, la figure 1(b) représente le DataGuide de l'arbre de la figure 1(a) (qui représente l'arbre de la collection). Par exemple le chemin d'étiquète *livre/chapitre/section/para* apparait deux fois dans la figure 1(a), par contre dans la DataGuide (figure 1(b)) il n'est représenté qu'une seul fois.

Dans le cas où les nœuds de l'arbre de la collection ne sont pas identifiés (aucune méthode d'identification n'est utilisée), le DataGuide ne peut nous renseigner que si un chemin existe dans l'arbre de la collection sans pouvoir autant l'identifier d'une manière unique dans l'arbre de la collection (c'est-à-dire dans quel document apparait-il).

[1]L'arbre de la collection :est de prendre tous les représentations en arbres des documents de la collection est de les relier tous à une seule racine.



#0	#1	#2	#3	#4	#5	#6
&0	&1	&2	&3	&4, &7	&5, &6	&8

(c) Table des annotations

Index	XML	Résumé
&2, &5, &6, &8	&2	&8

(d) table de contenu

Figure 3.1 – schéma explicatif du DataGuide

Alors une identification des nœuds du DataGuide et ceux de l'arbre de la collection est nécessaire pour pouvoir identifier les nœuds de la collection à partir de DataGuide d'une manière unique, cela se fait en stockant les identifiants dans une table nommée table des annotations qui est composée de deux lignes, la ligne d'en haut contient les identifiants de DataGuide et celle d'en bas contient les identifiants de l'arbre de la collection, l'identifiant de DataGuide pointe vers les identifiants des nœuds de la collection qu'il représente. Comme montré dans la figure1(c), en prenant exemple du nœud *para* de DataGuide identifié par l'identifiant #5 accédé par le chemin d'étiquète *livre/chapitre/section/para* pointe vers les identifiants &5, &6 de la collection accédée par ce même chemin d'étiquète.

Avec le DataGuide et la table d'annotation on indexe toute la structure des documents de la collection avec les relations hiérarchiques, mais il reste le manque d'identification précise des relations *père/fils* entre les nœuds des documents de la collection. (Par exemple : on sait d'après le DataGuide que les nœuds portant le label *para* sont les fils des nœuds *section* (relation hiérarchique) mais qu'on ne peut identifier depuis le DataGuide avec précision (la relation *père/fils*) que le nœud *para* identifier par &5 et le fils de nœud section *identifier* par &4).

Le contenu textuel quant à lui est sauvegardé dans un fichier inverse, où chaque terme est associé à l'identifiant des nœuds de la collection qui le contiennent et le tout est enregistré dans la *table de contenu* figure1 (d).

L'absence des liens hiérarchiques père/fils de différents niveaux entre les nœuds comme énoncé ci-dessus crée des confusions lors de la recherche, par exemple si on cherche à retourner les nœuds(leur chemin d'accès dans la collection) qui contient le mot clé *index,* on le recherche d'abord dans la *table de contenu* et on s'en aperçoit qu'il est contenu dans les nœuds identifiés par les identifiants suivants (&2, &5, &8, &6), pour chaque identifiant trouvé on doit construire son chemin d'accès, en prenant exemple de l'identifiant &6, on va aller à la table des *annotations*, on trouve qu'il est associé au nœud *#6* qui dénote l'étiquette *figure* de DataGuide, et pour accéder à ce nœuds dans le DataGuide on doit passer par le chemin d'étiquète (*livre/chapitre/section/figure*) et si on représente ce chemin par leur identifiant on aura le chemin *#0/#3/#4/#6* et d'après la *table des annotation* on peut construire deux chemins d'identifiant qui sont *&0/&3/&4/&6* et *&0/&3/&7/&6* sauf que le deuxième n'existe pas dans l'arbre de la collection figure1(a), alors il y a risque de retourner des chemins inexistants lors de la recherche.

Dans la section suivante nous présentons une approche proposée par Fouad Dahak '*Un DataGuide annoté avec un index de contenu*' qui solutionne le problème posé ci-dessus où on a apporté quelques modifications qui permettent d'identifier les nœuds d'une manière unique sans confusion.

III.3 Approche implémentée: Un DataGuide annoté avec un index de contenu.

Une approche d'indexation des documents semi-structurés doit indexer la structure et le contenu et faire le lien entre les deux structures d'index, l'approche consiste à indexer la structure avec un DataGuide et de représenter l'index du contenu dans un fichier inverse étendu en réalisant un lien virtuel entre les deux garce à une signature virtuelle, cette signature virtuelle nous permet de retourner le nœud où le terme qui apparait avec son poids, cette valeur est nécessaire pour trier les résultats selon l'ordre de pertinence. Notre index se base donc sur un DataGuide annoté avec un fichier inverse étendu. Quant au modèle de recherche utilisé, c'est le modèle vectoriel étendu de [karen, 2005] présenté dans le chapitre précèdent. Dans ce qui suit nous détaillerons dans les sections les différentes étapes de l'approche.

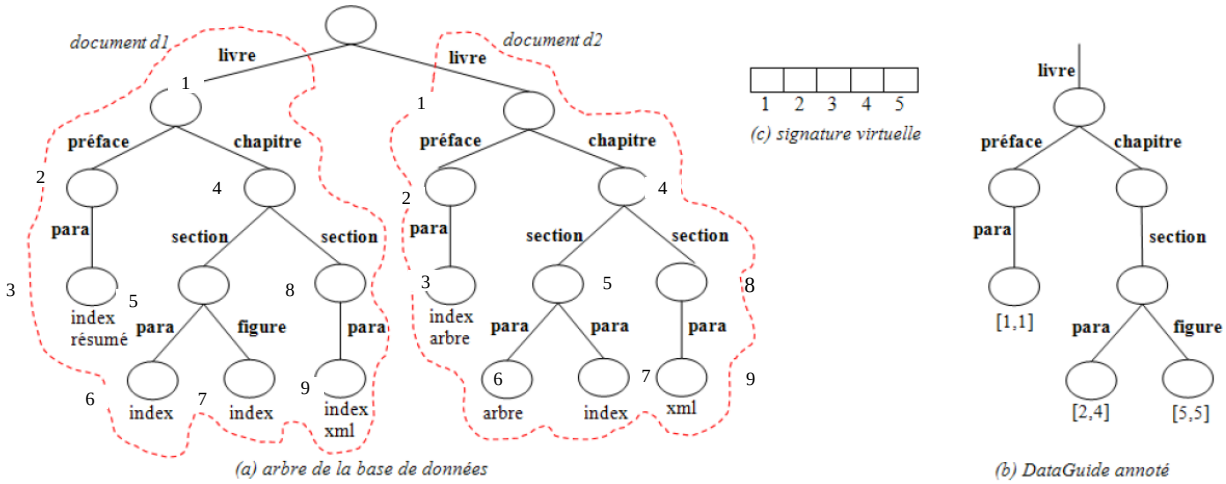


Figure 3.2

III.3.1 Le DataGuide annoté :

Comme défini précédemment, le DataGuide est un résumé de chemin d'étiquète construit sur la base de l'arbre de la collection, où chaque chemin ne figure qu'une seul fois, la construction de DataGuide se fait de la manière suivante :

- Pour chaque document on construit sa représentation en arbre.
- On lie tous les représentations d'arbre des documents à une seule racine, l'arbre résultant nous le nommons l'arbre de la collection (AC) figure 2(a).
- On crée le DataGuide de l'arbre de la collection.
    - La figure 2 (b) représente le DataGuide annoté de l'arbre de la collection de la figure 2(a) formé des deux documents d1 et d2, le chemin *livre/chapitre/préface* figure deux fois dans l'arbre de la collection, mais dans le DataGuide ne figure qu'une seule fois.
- Chaque nœud N_c de la collection est représenté par un nœud N_{dg} de DataGuide, c'est le nœud N_{dg} accéder par le même chemin dans le DG que celui accédé par le nœud N_c dans l'AC.
    - par exemple : le nœud *para* accédé par le chemin *livre/préface/para* de la figure2(b) représente tous les nœuds *para* accédés par ce même chemin de la figure2(a)).
- Les nœuds 'N_{dga}' qui doivent être annotés du DG sont les nœuds N_{dg} du DG qui représentent les nœuds N_c de l'AC qui sont soit des nœuds texte ou soit des nœuds mixtes.
    - par exemple : (le nœud *para* accédé par le chemin *livre/préface/para* de la figure2(b) est annoté parce qu'il représente des nœuds contenant du contenu textuel dans l'arbre de la collection figure1(a)).

L'annotation des nœuds du DG se fait en deux étapes :

- La première étape de l'annotation consiste à attribuer à chaque nœud annoté N_{dga} de DG une valeur que l'on nomme '*max'*, cette valeur 'max' est le nombre maximum de fois du chemin menant au nœud N_{dga} c'est répéter dans un document donner dans la collection (dans l'AC pour un document donné).
    - Par exemple pour le chemin *livre/chapitre/section/para* apparait deux fois dans *d1* et trois fois dans *d2*, alors le nœud *para* de DataGuide accessible par le chemin précédent on lui assigne la valeur 3 (car 3 c'est le nombre maximal de fois que le chemin menant à ce nœud est répété dans un document de la collection et c'est le document d2).
- La deuxième étape de l'annotation consiste à construire pour chaque nœud N_{dga} de DG un intervalle [*minpos, maxpos*], la construction des intervalles se fait en parcourant le DG de haut en bas, de gauche à droite d'une manière récursive (parcoure préordre), et pour chaque nœud N_{dga} rencontré son intervalle est construit de la manière suivante :
    - *Minpos :* la somme de tous les valeurs '*max'* des nœuds N_{dga} qui le précède + la valeur '1'.
    - *Maxpos* : la valeur '*minpos'*+ la valeur '*max'* de nœud – la valeur '1'.

Donc on aura des intervalles qui se succèdent les uns aux autres, par exemple .

- *Par exemple* : dans la figure 2(b) le nœud *para* accessible par le chemin *livre*/*préface*/*para* est annoté par [1,1] car il est le premier nœud rencontré en parcourant préordre et son *max* est égale '1', puis le nœud *para* accessible par le chemin *livre/chapitre/section/para* est annoté par [2,4] car sa valeur *max* est de '3' est la somme des *max* précédents est de 1 donc *minpos* est 1+1 = 2 et sa valeur *maxpos* est de 2+3-1 = 4 et ainsi de suite.

III.3.2 La signature virtuelle :

La signature virtuelle permet de relier l'index de la structure à celui du contenu, elle est construite au monument de l'indexation et lors de la recherche, la signature virtuelle nous permet d'identifier les nœuds où les termes sont apparus dans les documents d'une manière exacte avec leur poids.

La signature virtuelle est une suite de bits, ça taille est la somme de tous les valeurs '*max'* des nœuds N_{dga} du DG, cette valeur représente le nombre de nœuds feuilles de la collection. pour chaque nœud N_{dga} du DG une suite successive de bits lui est attribuée, sa valeurs *minpos* dénote la postions du premier bit assigné dans la signature virtuelle et la valeur *maxpos* dénote la position de dernier bit qui lui est assigné. Par exemple : dans la figure 2(b) le nœud accessible par le chemin *livre/chapitre/section/para* est annoté par [2,4] signifiant que les bits 2, 3 et 4 de la signature représentent ces nœuds.

Formule du calcule de la taille de la signature virtuelle :

- $Length(s) = \sum_{i=1}^{ndg} l_i = \sum_{i=1}^{ndb} max_{j=1}^{nbd}(\text{Nb}(c_i,d_j))$. Formule 3.1

  Où

  - ndg est le nombre de nœuds annotés.
  - l est la taille de l'intervalle du nœud annoté.
  - ndb est le nombre de nœuds contenant du texte.
  - nbd est le nombre de documents de la collection.
  - Nb renvoie le nombre d'apparitions du chemin d'étiquète c dans le document d.
  - max renvoie le maximum de Nb de tous les documents de la collection.

Pour chaque nœud texte ou mixte N_c de la collection dans un document on lui correspond un bit dans la signature virtuelle, l'attribution de ce bit se fait de la manière suivante :On calcule l'ordre du chemin d'étiquète menant au nœud N_c par rapport au même chemin d'étiquète dans le document en partant de gauche à droite plus (+) la valeur *minpos* du nœud N_{dga} du DG qui est accédé par le même chemin d'étiquète que celui accédé par le nœud N_c de la collection mois la valeur '1'(car l'annotation commence de 1), et de cette manière on obtient la position du bit dans la signature virtuelle qui correspond à un nœud donné N_c de la collection. Par exemple : pour le premier nœud *para* de document 2 accessible par le chemin '*p*'=*livre/chapitre/section/para*, c'est le deuxième bit de la signature virtuelle qui lui correspond car la valeur *minpos* de nœud *para* dans le DataGuide accessible par le même chemin est de '2' plus l'ordre de ce nœud qui est à '1' dans le document moins (–) la valeur '1' ça nous donne la valeur '2' alors pour le deuxième nœud *para* accessible par le même chemin *p* est le troisième bit de la signature virtuelle et ainsi de suite.

III.3.3 Fichier inverse étendu

Le fichier inverse étendu stocke l'index du contenu de la manière suivante :

pour chaque mot clé de chaque document, on lui construit la signature virtuelle en mettant à '1' les bits représentant les nœuds N_c de la collection où il apparait dans le document et à '0' les autres bits, par exemple le mot clé *arbre* dans le document d_2 on le représente par la ligne suivante *arbre*→ (d2, 11000), signifiant que dans le document d_2, par le premier bit qui est à '1' le mot clé *arbre* apparait dans le premier nœud accessible par le chemin *livre/préface/para*(car le nœud *para* accessible par ce chemin occupe dans la signature virtuelle le premier bit), et par le deuxième bit qui est à '1' le mot clé apparait dans le premier

nœud accessible par le chemin *livre/chapitre/section/para*(car le nœud *para* accessible par ce chemin occupe dans la signature virtuelle les bit de 2 à 4). Alors pour trouver la position du nœud correspondant dans le document de la collection on fait ce calcule suivant:

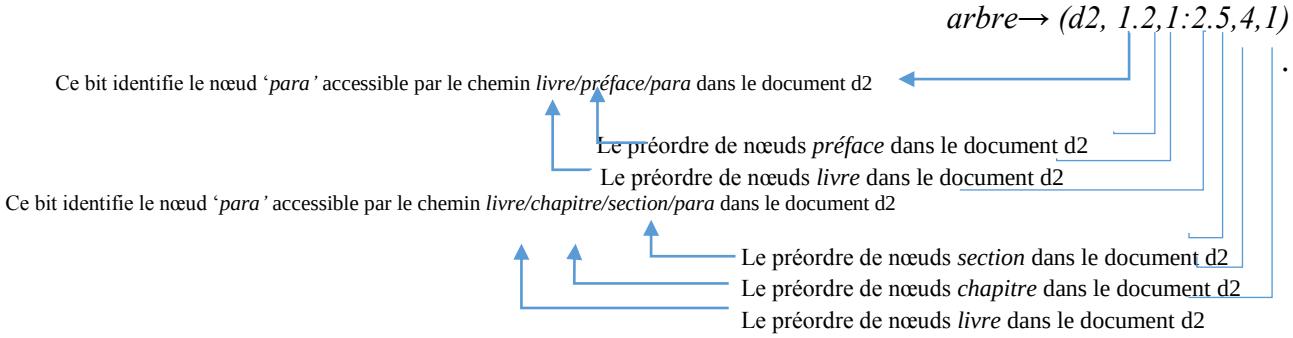
- on prend dans la signature virtuelle la position de bit qu'est à '1' que nous nommons 'p' moins la valeur *minpos* de l'intervalle contenant la valeur de la position 'p' dans le DataGuide puis on ajoute la valeur '1' (car on commence la numérotation des bits de la signature virtuelle à partir de 1) donc la formule est *p-minpos+1* (tel que p ∈ [minpos, maxpos]); alors dans notre cas on prenant exemple le deuxième bit qui est à un 1, ça position 'p' est à 2, qu'elle est contenue dans l'intervalle [2,4], donc sa position dans le document d2 est 2-2+1 =1 ; le nœuds annoté par l'intervalle [2,4] est le nœud *para* accessible par le chemin *livre/chapitre/section/para*, donc on va prendre le premier nœud *para* accessible par ce chemin dans le document d2, et de cet façon on pourra identifier tout les nœud feuille de la collection.

La signature virtuelle est construite pour chaque mot clé pour chaque document où ce mot clé apparait. Or souvent que quelque bit de la signature virtuelle qui sont mis à '1' vu qu'un terme n'apparait pas dans tous les nœuds d'un document, c'est pour ça au lieu de sauvegarder la signature virtuelle tout entière on sauvegarde que les positions des bits qui sont à '1'.donc l'exemple précédent devient *arbre→ (d2, 1:2) les nombre 1 et 2 représente les position des bit qui sont à1 dans la signature.*

Remarque : les mots clés de l'index de contenu des nœuds feuille son extrait avec la méthode de l'indexation classique explique dans le chapitre I à la section 2.1 (tokanisation, lemmatisation, élimination des mots vides).

III.3.4 La modification apportée :

Avec la signature virtuelle on identifie avec précision où le terme apparait dans le document, mais il n'existe aucun moyen de pouvoir identifier son père (car aucune méthode d'identification n'est utilisé pour identifier les nœuds internes), or il faut qu'on puisse les identifier pour pouvoir propager les poids des termes ou le score de pertinence vers leur ancêtre et les retourner (les nœuds interne) aux utilisateurs. Donc nous avons apporté une modification qui consiste à ajouter à la position des bits de la signature virtuelle les identifiants des ancêtres de nœuds que ce bit de la signature virtuelle identifie dans le document, l'exemple précédent devient :



Remarque :

- dans cet exemple on a utilisé la numérotation préordre pour identifier les nœuds dans les documents.
- La position de bit de la signature virtuelle est séparer par une point de la liste de préordre des ancêtres, et les préordres des ancêtres son séparer par des virgule.
- Dans le fichier inverse or mit la position de bit et la liste des préordres on peut ajouter le poids du terme dans le nœud.
- Concernant la liste des preordres ajouté dans la modification, on peut ôter le preordre de la racine, car par défaut il est toujours à '1' et il est l'ancêtre de tout les nœuds de document (toujours présent implicitement).

Cette modification ajouter résout le problème de la propagation des scores lors de l'appariement (l'algorithme 1 page 56 montre la façon de propager le score).

Le poids de terme dans les nœuds feuilles est calculé par la formule suivante :

$$W_{i,j}=tf_{ij}*ief_j=\frac{nbocc(t_i,f_j)}{nbterm(f_i)} * \log(\frac{|F_c|}{|nf_j|})/\log(|F_c|). \quad \text{Formule 3.2}$$

- $nbocc(t_i,f_j)$ est le nombre d'occurrences de terme t_i dans le nœud feuille f_j.
- $nbterm(f_i)$ est le nombre de termes dans le nœud feuille.
- $|F_c|$ est le nombre de nœuds feuilles de la collection.
- nf_j est le nombre de nœuds feuilles contenant le terme t_i.

Le modèle de recherche utilisé est le modèle vectoriel étendu proposé par [Sauvagnat ; 2005] : la pertinence d'un nœud feuille par apport à une requête est évaluer selon la formule suivante :

$$RSV(Q, n_f) = \sum_{i=1}^{n} w_i^q * w_i^{nf} = \sum_{i=1}^{n} tf_i^q * ief_i^2 * tf_i^{nf} \quad \text{Formule 3.3}$$

- tf_i^q : est la fréquence de terme i dans la requête
- tf_i^{nf} : est la fréquence de terme i dans le nœud feuille nf.
- ief_i^2: est la fréquence inverse élément de terme i.

La propagation de la pertinence :

La pertinence des termes dans les nœuds internes est calculée lors de l'appariement, le score des nœuds feuilles est propagé vers le haut en prenant en compte deux facteurs , le premier est la distance entre le nœud interne (l'ancêtre) et le nœud feuille, le deuxième est la longueur des nœuds feuilles et leurs longueurs moyennes.

La formule de propagation de la pertinence :

$$I_n = \sum_{k=1..N} \alpha^{dist(n,nf_k)-1} * \beta * RSV(q, nf_k) \quad avec \begin{cases} \beta = & {}^{l_k}/_{\Delta l} \text{ si dist(n, nf}_k\text{) = l et l}_k < \Delta l \\ & \log({}^{l_k}/_{\Delta l}) \text{ si dist(n, nf}_k\text{) = l et} \\ & \\ & 1 \text{ sinon} \end{cases}$$

Formule 3.4

Où :

- I_n le nœud interne.
- dist(n, nf_k) la distance entre le nœud i_n et le nœud feuille nf_k.
- l_k la longueur de nœud feuille.
- Δl la longueur moyenne des nœuds feuilles.

Algorithme 1:

Procédure de recherche d'une requête orientée contenue :
Début
Initialiser le résultat = ensemble vide.
Δl représente la Taille moyen des nœuds feuilles
Pour chaque mot clé t de la requête faire
Début
Calculer le poids de mot clé dans la requête w_q.
Chercher le mot clé *t* dans le fichier inverse et récupérer les couples (docid[2], nodelist[3]) correspondant.
Pour chaque couple retourné faire
Début
Pour chaque couple (nodepos[4], listpreordre[5], w_n[6], l[7]) présent dans *nodelist* faire
Début

- Extraire dans le dataguide annoté le chemin d'étiquette *c* menant au nœud N_{dga} annoté contenant l'intervalle comportant *nodepos* et la valeur *minpos* de cet intervalle.
- Calculer la position *posca* du nœud dans l'arbre de la collection accessible par le chemin *c* avec *posca = nodepos – minpos +1*.
- Calculer le score de pertinence *pt* du terme dans le nœud avec $pt = w_n * w_q$.
- Si la ligne (docid, c, posac, p[8]) existe dans le résultat alors ajouter le score de pertinence *pt* du terme dans le score de pertinence *p* de nœud avec *p=p+pt*
- Sinon ajouter la ligne (docid, c, posac, pt) dans le résultat.
- Initialiser *dist*a zéro

Pour chaque *preordre* de *listpreordre* de gauche à droite faire
Début

- Incrémenter dist
- Propager le score de pertinence *pt* au nœud interne par la formule suivant.
- $pni = 0.8^{dist-1} * \beta^{9} * pt$
- Si la ligne (*docid, preordre, p*) existe dans le résultat ajouter le score de pertinence du terme *pt* dans le score de pertinence *p* de nœud avec *p=p+pni*
- Sinon ajouter la ligne (*docid, preordre, pni*)

---

[2]Identifiant du document.
[3]Nodelist est compose de la liste des nœuds contenue le terme
[4]Nodepos la position dan la signature virtuelle
[5]Liste des préordre des ancêtres de nœud.
[6]Le poids dans le nœud.
[7]Taille de nœud
[8]Score de pertinence de nœud par apport a la requête
[9]β est calculé selon la formule 2, on lui passe les paramètre Δl et l

Fin.

Fin.

Fin.

Astuce pour amélioration de la méthode :

Dans le fichier inverse où on a apporté notre modification, au lieu d'ajouter toute une liste de preordre ancêtre a coté de bit de position de la signature virtuelle, on ajoute seulement que le preordre du premier ancêtre (le père) est les autre seront stoker dans une table où on pourrait les récupérer par la suite. Cette technique nous permet de réduire l'espace de stockage de l'index. La table est former de deux colonne la premier colonne contient les preordre de l'arbre du document, la deuxième les preordre père comme montrer dans figure 3.3.

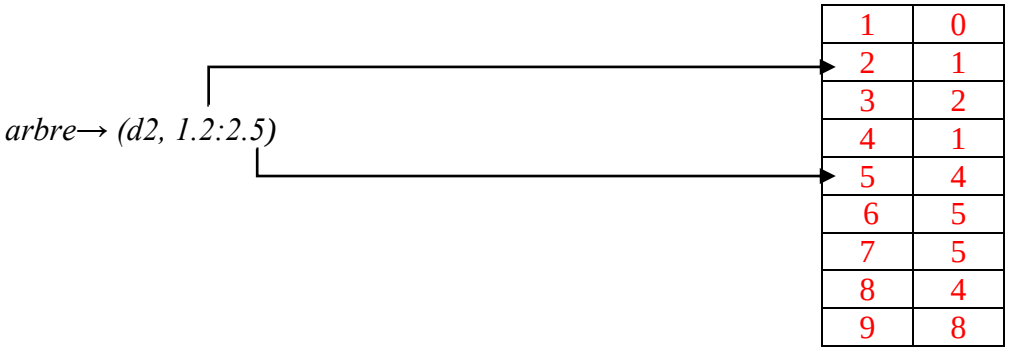


Figure 3.3 table de preordre du document d1 de la figure 3.2.

L'inconvénient de la méthode est que pour tout ajout d'un nouveau nœud y a risque de recalculer la position des bits de la signature virtuelle pour chaque mot clé dans le fichier inverse, cela peut se produire si on ajoute dans un document donner un nœud *n* accessible via un chemin *p* existant déjà dans ce document *d*, et ce document *d* contient le nombre maximum de fois que ce chemin *p* est répété dans la collection, alors il cause un décalage de la signature virtuelle depuis la position *maxpos* de nœud annoté N_{dga} de DataGuide accessible via le chemin *p* jusqu'au dernier bit de la signature virtuelle, pour remédier à cela, on utilise des intervalles large lors de l'indexation en rajoutant un nombre α de bits à l'intervalle d'origine, le paramètre α doit faire un objet d'étude pour trouver sa valeur idéale.

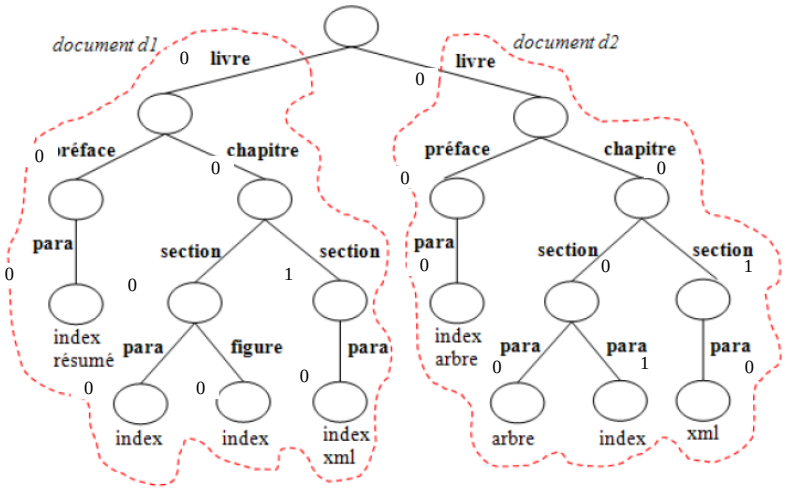
III.4 Proposition d'une autre solution :

La modification apporter à l'approche ne résout le problème que partiellement, elle ne permet que la possibilité de propagé le score des termes vers les nœuds interne lors de la recherche.

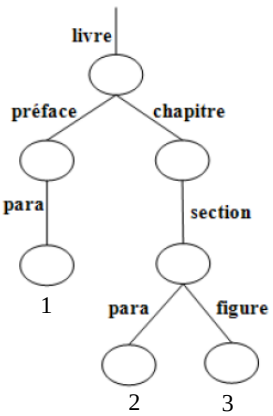
Nous proposant dans ce qui suit une solution qui permet d'indexer aussi bien les nœud feuille et les nœud interne, la structure de l'index consiste a créé deux DataGuides annoté un pour l'arbre de la collection et un autre pour les arbres numérotés des documents de la

collection puis relier les deux DataGuides annotées au fichier inverse, la solution se fait de la manière suivante :

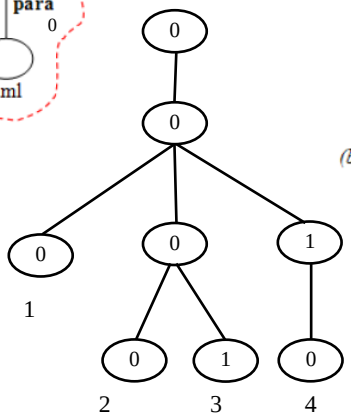
- on crée le DataGuide de l'arbre de la collection est on l'annote avec la numérotation préordre mais que pour les nœuds feuilles.
- Le deuxième DataGuide est construit de la manière suivant :
  - Pour chaque document de la collection en crée sa représentation d'arbre.
  - On identifie chaque nœud de l'arbre avec la méthode d'identification '*sibling numberse*' puis on relier tous les arbres numéroté à une seule racine.
  - on crée le DataGuide de l'arbre résultant (figure 3(d)) est on l'annote avec la numérotation préordre mais aussi que pour les nœuds feuilles.



(a) l'arbre de la collection numéroté avec l'identification '*Sibling numberse*'



(b) DataGuide annoté



(c) DataGuide annoté de l'arbre numéroté de l'arbre de la collection

- Pour identifier un nœud feuille *nf* dans le document *D*, on lui associer l'identifiant (préordre) de nœud feuille *ndg* de chaque DataGuide accéder par le même chemin que le nœud feuille *nf* dans le document *D*.
  - Par exemple : l'identifiant de nœud '*para'* situé à la troisième position en partant de la gauche dans le document d_2 est (d_2, 2,3), le *2* fait référence au nœud *para* accéder par le chemin *livre/chapitre/section/para* (c'est le même chemin d'étiquète de nœud dans le document extrait de premier DataGuide)et le *3* fait

référence au chemin *0/0/0/1*(c'est le même chemin d'identification de nœud dans le document extrait de deuxième DataGuide),et on combinant les deux en aura ce chemin *livre[0]/chapitre[0]/section[0]/para[1]*. de cet maniéré on peut extraire le chemin XPath accélérateur de mot clé dans le nœud de document où il apparait.

- Pour identifier un nœud interne *n* dans un document D, on prend le nœud feuille *nf* fils de nœud *n* qui est situé le plus à gauche dans l'arbre de document, puis on identifier ce nœud feuille *nf* comme explique ci-dessus, on ajoute après à l'identification un nombre qui désigne le nombre de niveau séparent entre le nœud interne *n* et son nœud descendent feuille *nf*.
    - Par exemple pour identifier le nœud interne *chapitre* dans le document d_2, on précède d'abord à l'identification de son nœud descendent feuille situé le plus à gauche, dans notre exemple c'est le deuxième nœud feuille *para* sont identifiant est (d_2, 2, 2) puis on ajoute à l'identification le nombre 2 car le nombre de niveau séparent entre le nœud interne *chapitre* et le nœud feuille fils *para* est de 2, donc on aura à la fin la ligne suivante (d_2, 2, 2, 2).
- Avec cette identifiant on peut extraire le chemin XPath accélérateur de nœud interne, pour notre exemple précédent on procède comme suit :
    - Le premier '2' de la ligne (d_2, **2**, 2, 2) fait référence au chemin *livre/chapitre/section/para.* Le deuxième '2' (d_2, 2, **2**, 2) fait référence au chemin d'identifiant *0/0/0/0*, puis en combinant les deux on aura *livre [0]/chapitre [0]/section [0]/para [0]*, quand au dernier '2' (d_2, 2, 2, **2**) il nous indique combien d'étiquète doit on extraire du chemin précédent a partir de la droite pour avoir le chemin finale d'accès au nœud interne, donc en aura a la fin ce chemin *livre[0]/chapitre[0]* (*livre [0]/chapitre [0]/**section [0]/para [0]** c'est le chemin en gras qui a était effacer*).

Cette méthode nous permet de résoudre le problème d'identification des nœuds feuille et interne d'une manière unique, mais pour pouvoir identifier les relations hiérarchique (ancêtre/descendent) entre les nœuds il faut d'abord restituer les chemins de chaque nœud puis faire une comparaison entre eux. Si le chemin d'un nœud est inclus dans l'autre en partant de gauche à droite alors ce nœud est l'ancêtre de l'autre, pour l'exemple précédent le chemin *livre [0]/chapitre[0]* du nœud *chapitre* est inclus dans le chemin ***livre [0]/chapitre [0]****/section [0]/para [0]* du nœud *para* où on le voit en gras, donc le nœud *chapitre* est l'ancêtre de nœud *par.*

Perspective :

Nous projetons d'améliorer l'approche proposée on essayant de trouver une méthode qui permet de rendre l'identification des relations hiérarchique entre les nœuds avec une manière plus souple.

III.5 Conclusion :

Dans ce chapitre on a présenté une approche d'indexation 'un DataGuide annoté avec un fichier inverse étendue' où on a apporté des modifications au niveau de fichier inverse pour pouvoir propage les score des nœuds feuilles vers leur ancêtre, l'approcher présenter on la implémenté avec le modèle vectorielle étendue proposé par [Sauvagnat, 05], pour approuver son efficacité.

Toute fois même la modification apportée ne remédier pas à tous les inconvénients de l'approche, comme celle de pouvoir identifier les nœuds internes avec leur chemin d'accès, alors on a proposé une solution avec deux DataGuide, un DataGuide pour indexe les noms des éléments l'autre pour indexer les identifier des éléments des nœuds arbre de document XML.

Dans le chapitre suivant on présente les résultats de l'implémentation de l'approche sur la collection INEX 2009.

Chapitre 4
Évaluation et résultat

Introduction :

Nous présentons dans ce chapitre l'environnent de développement et les déférents outilles utiliser pour la réalisation, l'implémentation et l'évaluation dans notre approche ensuite nous montrons nous résultat et teste.

IV.1 Outils et développement :

IV.1.1 Java :

Java est un langage de programmation moderne développé par **Sun Microsystems** (aujourd'hui racheté par **Oracle** C'est un langage de programmation orienté objet. Il permet de créer des logiciels compatibles avec de nombreux systèmes d'exploitation (Windows, Linux, Macintosh, Solaris). Java donne aussi la possibilité de développer des programmes pour téléphones portables et assistants personnels. Enfin, ce langage peut être utilisé sur internet pour des petites applications intégrées à la page web (applet) ou encore comme langage serveur (jsp).

IV.1.2 MySQL :

MySQL (*StructuredQueryLanguage*) Système de gestion de bases de données relationnelles (SGBDR) sous licence GNU très utilisé pour mettre en ligne des bases de données. Stocke les données dans des tables séparées plutôt que de tout rassembler dans une seule table. Cela améliore la rapidité et la souplesse de l'ensemble. Les tables sont reliées par des relations définies, qui rendent possible la combinaison de données entre plusieurs tables durant une requête.

MySQL fonctionne sur beaucoup de plates-formes différentes, incluant AIX, BSDi, FreeBSD, HP-UX, Linux, Mac OS X, NetBSD, OpenBSD, OS/2 Warp, SGI Irix, Solaris, SunOS, SCO OpenServer, SCO UnixWare, Tru64 Unix, Windows 95, 98, NT, 2000 et XP.

IV.1.3 Api SAX xerces:

L'api SAX xerces est utilisé pour parser les documents XML, elle nous permet de construire l'arbre de document XML et d'extraire le contenue.

IV.1.4 Eclipse IDE :

Eclipse *IDE* *(Integrated DevelopmentEnvironment*) c'est un logiciel qui simplifie la programmation en proposant un certain nombre de raccourcis et d'aide à la programmation. Il est développé par IBM, est gratuit et disponible pour la plupart des systèmes d'exploitation.

IV.1.5 JDBC :

JDBC (Java DataBaseConnectivity) désigne une API pour permettre un accès aux bases de données avec Java.

IV.2 Test et Résultats :

Nous expérimentations ont été portés sur 4803 document de la collection INEX 2009 avec une taille de 200 MO répondant à 68 requêtes.

Nous utilisons la collecte, les requêtes fournies par INEX et notre logiciel pour produire les éléments XML pertinents vis-à-vis de notre système, les résultats retournés sont évalués en utilisant les mesures d'évaluation de pertinence fournis par INEX.

Dans ce qui suit en présente les diffèrent tests portés sur notre approche évaluant l'espace mémoire utilisé, le temps d'exécution des requêtes et la fiabilité d'un système base sur notre index de stockage.

IV.2.1 Test mené sur la taille de l'index :

Dans cette section, nous avons comparés la taille de l'index de notre approche avec une approche d'indexation basé sur les chemins XPath Accélérateur. Le tableau 4.1 montre la comparaison entre les deux approches.

	Taille de l'index	Taille index/taille de la collection	Taille de la collection
indexation basé sur les chemins	438 MO	219%	200 MO
notre approche d'indexation	127 MO	63.5%	
Taille : notre approche d'indexation/ indexation basé sur les chemins	29%		
Le gain[10]	71%		

Tableau 4.1 – comparaison des tailles des indexes.

[10] Gain : (taille (index base sur les chemins) – taille (notre approche))/taille (index base sur les chemins).

On constate que notre index économise bien de l'espace mémoire car il présente une taille inférieure à celle de l'index basé sur le chemin et sur la taille de la collection.

IV.2.2 Evaluation de l'approche de l'indexation avec un modelé de recherche :

Dans cette section, on a implémenté notre approche d'indexation avec un modèle de recherche vectorielle (présenter dans le chapitre 3) et on la comparer avec le même system mais sa structure de stockage est basé sur l'index de chemin, pour pouvoir montre que notre index peut servir de structure de stockage des index pour un système sans que ce dernier ne perd pas de son efficacité.

La figure 4.1 montre le résultat de l'évaluation de notre système dans la tache Thorough Task de INEX 2009 sur 33 requêtes et la Figure 4.2 montre les résultats de la tâche Thorough Task pour le système base sur l'index des chemin XPath accélérateur.

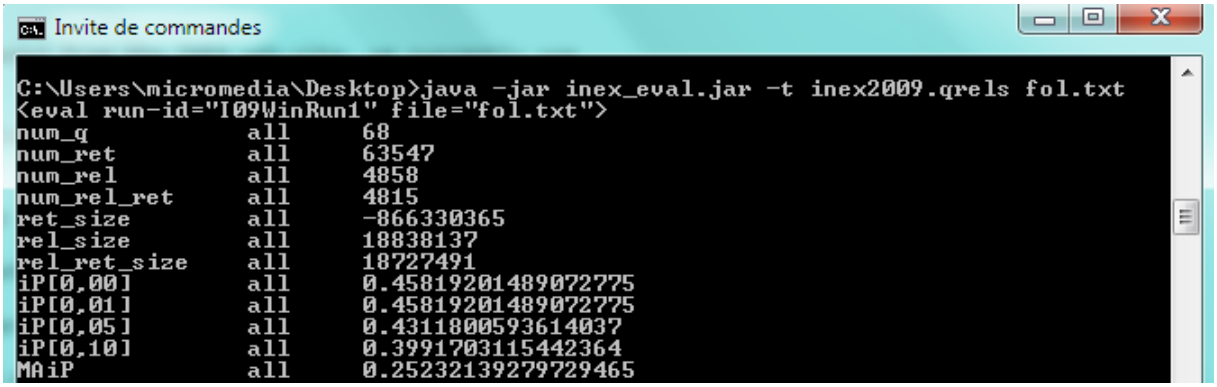


Figure 4.1 – résultat de la tâche Thorough Task pour notre approche.

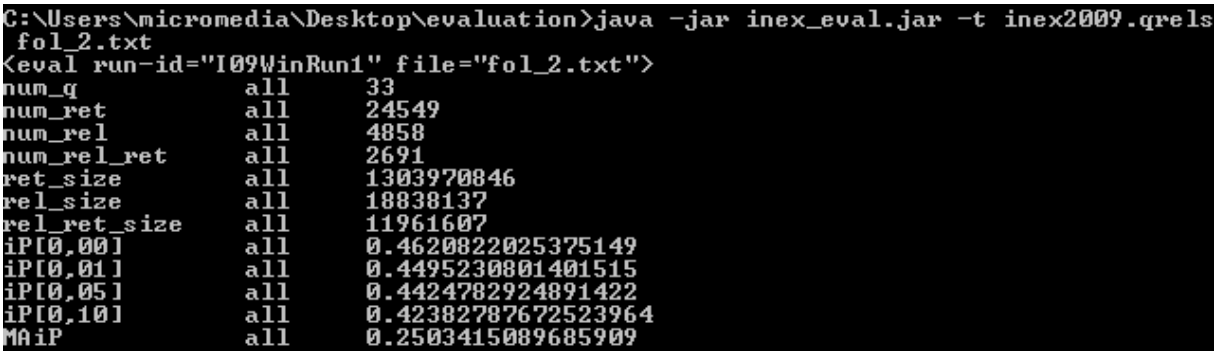


Figure 4.1 – résultat de la tâche Thorough Task pour le system basé sur l'index XPath Accelerateur.

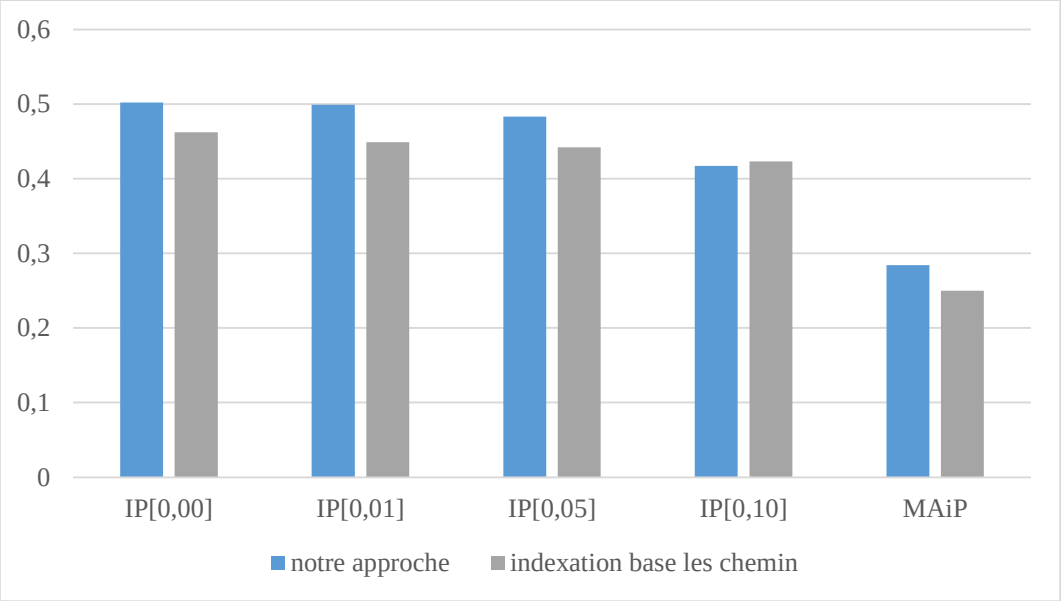


Figure 4.3 – histogramme de comparaison entre les deux systèmes.

D'après l'histogramme en constate que le système basé sur notre structure d'index ne perd pas de son efficacité, ce qui démontre que notre approche d'indexation respect bien les critères d'un bon index c'est-à-dire qu'elle index le contenue, la structure et préserve le lien entre les deux.

IV.2.3 Test sur le temps d'exécution :

Dans cette section on compare le temps d'exécution mit par le Système basé sur notre approche pour répondre aux 33 requête par apport au système basé sur l'indexation par chemin.

Temps d'exécution des 33 requêtes

Système base sur notre approche	3 min
Système basé sur l'indexation par chemin	1 h : 20 min

On constate que le système basé sur notre approche répond plus rapidement aux requêtes que le système basé sur l'index de chemin.

Conclusion

Dans ce chapitre nous avons présenté l'environnement technique de notre développement et les résultats de nos expérimentations qui montrent que la structure d'index répond bien aux exigences d'un bon indexe dans les documents semi-structure (l'espace mémoire utilisé, temps d'exécution..).

Conclusion générale

Conclusion générale

Notre objectif dans ce mémoire est d'étudier les différentes approches d'indexation des documents semi-structurés de type XML, puis d'implémenter une approche d'indexation « physique » (ou structure d'index) qui permet une indexation et une recherche efficace des informations dans de grandes collections de documents.

A l'issue de notre travail, nous avons proposé une approche d'indexation « physique » des documents XML basée sur l'utilisation conjointe d'un index de contenu (basé sur les fichiers inverses) et d'un index de structure (basé sur le DataGuide) tout en préservant un lien fort entre les deux index grâce à la signature virtuelle.

Notre approche a été implémentée, puis testée et évaluée sur la collection standard de RIS, la collection INEX 2009. Les résultats rapportés montrent bien que notre approche respecte tous les critères d'un bon index comme :

- Facilité de mise en œuvre
- index compact n'occupant que peu de place mémoire.
- rapidité dans l'exécution de la requête utilisateur (spécifiquement pour une requêté orientée contenu et structure).
- la manipulation d'une grande collection de document.

En perspectives, il serait intéressant d'aborder ces points :

- améliorer la structure de l'index au niveau de l'identification des nœuds interne.
- développer des techniques permettant d'identifier les relations hiérarchiques entre les nœuds de manière plus souple.
- Tester la structure d'index dans une plus grande collection afin de démontrer toute son utilité au niveau d'espèce mémoire économisé et la performance de retourné des résultats plus rapidement.

Enfin, ce travail nous a permis de nous familiariser avec le domaine de la RI, avec XML, avec la RIS et en particulier avec l'indexation. Outre de précieuses connaissances théoriques acquises, nous avons appris et maîtrisé un langage de programmation objet, le langage java avec lequel nous avons programmé.

Nous espérons enfin que ce travail puisse servir de base de travail pour les étudiants futurs qui s'intéresseraient à la RIS en général et à l'indexation des documents XML en particulier.

Bibliographie

[Azevedo, 2004] INEX'04 Proceedings of the Third international conference on Initiative for the Evaluation of XML Retrieval Pages 311-321

[Belew, 1989] Belew, R. K. *Adaptive information retrieval: Using a connectionist representation to retrieve and learn about documents*. In *ACM SIGIR Proceedings*. p. 11-20, 1989.

[Belkin, 92] N. J. Belkin and W. Croft. Information filtering and information retrieval: two sides of the same coin ? Communications of the ACM, 35(12).

[Ben-Aouicha, 2009] M. Ben-Aouicha. *Une approche algébrique pour la recherche d'information structurée*. Thèse de doctorat, Université Paul Sabatier, Toulouse, France, janvier 2009.

[Berry et al. 1999]. Berry, M. W., Z. Drmac, et E. R. Jessup : . *Matrices, vector spaces, and information retrieval. SIAM Rev.* 41(2), 335–362(1999).

[Boubekeur. 2008]. Boubekeur-Amirouche F. Contribution à la définition de modèles de recherche d'information flexibles basés sur les CP-Nets, thèse en informatique ,Université Toulouse III - Paul Sabatier .

[Boughanem, 2004.] M. Boughanem, W. Kraaij, and J.-Y. Nie. Modèles de langue pour la recherche d'information. In Les syst`emes de recherche d'informations. Hermes-Lavoisier, 2004.

[Boughanem, 2008] M. Boughanem and J. Savoy, editors. *Recherche d'information états deslieux et perspectives*. Hermès Science Publications, 2008.

[Cyril LAITANG, 2012] Impact de la structure des documents XML sur le processus d'appariement dans le contexte de la Recherche d'Information Semi-structurée, En vue de l'obtention du doctorat à l'université toulouse.

[Dahak, 2005] Indexation des documents Semi-Structurés, en vue d'obtention de diplôme magister.

[Dahak, 2008] Fouad Dahak, Boughanem, Balla. Indexation des documents XML : un DataGuide annotée avec un index de contenue

[Fuhr,03] N. Fuhr and K. Grossjohann. *XIRQL : a query language for information retrieval in XML documents*. In In Proceedings of SIGIR 2001, Toronto, Canada, 2003.

[Goldman,97] Goldman R and Widom J. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In Proceedings of the Twenty-Third International Conference on Very Large Data Bases (VLDB), 1997.

[kwak et al. 1989]. K.L. Kwok, a neural for probabilistic information retrivel. 12th international ACM SIGIR Conference on research and developpement in information retrival, pp 21-30, 1989.

[Hlaoua. 2007]. Hlaoua L. Reformulation de Requêtes par Réinjection de Pertinence dans les Documents Semi-Structurées, thèse en informatique , l'Université Toulouse III -Paul Sabatier .

[Hatano et al, 2002] k.Hatano, H.Kinutani, M.Yoshikawa, and S.Uemura. Infomation retreivale system for XML document 2002.

[Lalmas, 2006] M. Lalmas, G. Kazai, J. Kamps, J. Pehcevski, B. Piwowarski, and S. Robertson. INEX 2006 Evaluation Measures. In *INEX'06*,

[Lin . 1998]. Lin D. *An information-theoretic definition of similarity*. In Proc. 15th International Conf. on Learning, Morgan Kaufmann, San Francisco, CA, pp. 296–304 .

[Hlaoua, 2007] Hlaoua, S. « psychological relevance and information science », journale of the america society for information science.

[Maron et al, 1960] Maron, M and kuhns, j on relevance, probalistic indexing and information retrevale. Jornle of the association for computing machinery 7

[Mathias,02] Mathias G. Indexation et interrogation de chemins de lecture en contexte pour la Recherche d'Information Structurée sur le Web. Thèse pour obtenir le grade de Docteur de l'université JOSEPH FOURIER - GRENOBLE I, 2002.

[Malik et al. 2005]. Malik S., Kazai G., Lalmas M. , and Fuhr N. *Overview of INEX 2005. In Pre-Proceedings of the International Workshop of the Initiative for the Evaluation ofXML Retrieval, INEX 2005*, Dagstuhl Castle, Germany, November 28-30, 2005 pages 1–15.

[Mass et al. 2004]. Mass Y. and Mandelbrod M. *Component ranking and automatic query reffinement for XML retrieval. In INEX 2004 Workshop Proceedings*, pages 73,84.Dagsthul, Germany, December

2004.

[Morrison,80] Morrison D. R. PATRICIA: Practical Algorithm To Retrieve Information Coded In Alphanumeric. Journal of the ACM. 1968.

[Piwowarski,03] 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.

[Robertson, 1994] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In Proceedings of SIGIR

[Rocchio, 1971] J. Rocchio. *Relevance feedback in information retrieval*. 1971.

[Robertson, 1997] S.E. Robertson and S.Walker." on relevance weights whith little relevance information", in proceeding of the 20th annual internationnel ACM DIGIR conference on research and development in information retrieval.

[Sacks,94] Sacks-Davis R, Arnold-Moore T, and Zobel J. Database Systems for Structured Documents. In Proceedings of the International Symposium on Advanced Database Technologies and Their Integration (ADTI), 1994.

[Sauvagnat,05] Sauvagnat K. Modèle flexible pour la Recherche d'Information dans des corpus de documents semi-structurés, Thèse en vue de l'obtention du Doctorat de l'Université Paul Sabatier, 2005.

[Sauvagnat,06] Karen Sauvagnat,Mohand Boughanem, proposition pour la ponderation des termes et l'évaluation de la pertinance des élément en recherche 'information structurée, Actes de CORIA 2006.

[Sauvagnat, 05] K. Sauvagnat and M. Boughanem. A la recherche de noeuds informatifsdans des corpus de documents XML. In *Conférence francophone en Recherche d'Information et Applications, CORIA'05*, pages 119–134, 2005.

[Salton. 1970]. Salton G. *Automatic processing of foreign language document – Journal of the American Society for Information Science,* 21(3):187-194, May.

[Salton, 1983] G. Salton, E. Fox, and H. Wu. Extended boolean information retrieval. Communications of the ACM, 31(2) :1002–1036,

[Schileder,02] Schileder T and Meuss H. Querying and ranking XML documents.

[Rijsbergen, 79] J. van Rijsbergen. *Information retrieval*. 1979.

[W3C, 2005] Document object model(dom), rapport technique, 2005.

[Woods. 1997]. Woods, W. A.,. *Conceptual indexing : A better way to organize knowledge.Technical Report SMLI TR-97-61, Sun Microsystems Laboratories, Mountain View*,CA, April. ww.sun.com/research/techrep/1997/abstract-61.html.