Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Amélioration de la qualité des références fournies par les moteurs de recherche sur le Web

Nzamurambaho, Gaspard

Award date:
1999

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 04. May. 2026

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année académique 1998-1999

Amélioration de la qualité des références fournies par les moteurs de recherche sur le Web

Gaspard NZAMURAMBAHO

Mémoire présenté en vue
de l'obtention du grade de
Licencié en Informatique

Résumé

Ce travail essaie de répondre à la question de savoir si l'information que nous obtenons des systèmes de recherche d'information correspond à celle que nous souhaitons recevoir. Dans ce mémoire, l'analyse et la spécification d'une application pour l'amélioration de la qualité des références fournies par les moteurs de recherche sur le Web ont été réalisées. La reformulation de la requête pour affiner une recherche a été traitée par un programme qui proposera des nouveaux termes à ajouter ou à enlever dans la requête, et on pourra suivre, d'une manière automatique, l'apparition de nouveaux documents répondant à une requête donnée.

Abstract

The question is to know if the information we need is that we get. This thesis tries to answer to this question. In this thesis, the analysis and specification of software which improves the quality of the references retrieved by Web search engine have been realised. The query formulation problem is treated through a process known as *relevance feedback*. The user can follow automatically the evolution of new relevant items for his request.

Remerciement

Au terme de ce travail, je voudrais adresser mes remerciements à :

- *Madame le Professeur M. NOIRHOMME-FRAITURE de m'avoir permis de travailler sur un sujet qui m'intéressait et de tous ses précieux conseils;*
- *Monsieur L. GOFFINET pour sa grande disponibilité tout au long de la réalisation de ce travail;*
- *Monsieur J. VAN LOO de la sucrerie de Tirlemont pour sa franche collaboration;*
- *tout le personnel et étudiants de la 3ème Maîtrise de l'Institut d'Informatique à Namur pour leur aide technique;*
- *Ernest de m'avoir informé l'existence de la Licence en Informatique pour le cycle de deux ans à Namur;*
- *tous les amis, les parents, pour le soutien moral qu'ils ont su m'apporter.*

Enfin je voudrais signaler combien j'ai été très touché par l'esprit de compréhension et de contacts humains développé aux Facultés Universitaires Notre-Dame de la Paix à Namur, j'en suis reconnaissant.

Table des matières

TABLE DES MATIÈRES 1

CHAPITRE 1 : 3

1. INTRODUCTION GÉNÉRALE 3

CHAPITRE 2 : 5

2. MESURE DE L'EFFICACITÉ DE RECHERCHE D'INFORMATION 5

CHAPITRE 3 : 8

3. LES FACTEURS AFFECTANT L'EFFICACITÉ D'UNE RECHERCHE D'INFORMATION

3.1. INTRODUCTION 8
3.2. LES MÉTHODES D'INDEXATION 8
3.2.1. Indexation manuelle *10*
3.2.2. Indexation automatique *10*
3.3. FONCTIONS DE CORRESPONDANCE ET MESURES DE LA SIMILARITÉ 14
3.4. LES STRATÉGIES DE RECHERCHE D'INFORMATION 17
3.4.1. Introduction *17*
3.4.2. Recherche booléenne *17*
3.4.3. Recherche textuelle *18*
3.4.4. Recherche par les fonctions de correspondance *20*
3.4.5. Recherche séquentielle (serial search) *21*
3.4.6. Recherche dans les clusters *22*
3.4.7. Stratégie de relevance feedback *22*

CHAPITRE 4 : 29

4. MOTEURS DE RECHERCHE, MÉTA-MOTEURS ET WWW 29

4.1. INTRODUCTION 29
4.2. DÉFINITIONS 29
4.3. COMPARAISONS DES MÉTA-MOTEURS 30
4.3.1. Introduction *30*
4.3.2. Comparaisons *31*
4.3.3. Le nombre de documents retournés par *33*
les différents méta-moteurs *33*

5. AMÉLIORATION DE LA QUALITÉ DE RÉFÉRENCES FOURNIES PAR 35

LES *MÉTA MOTEURS* DE RECHERCHE SUR LE WEB 35

5.1. BUT DE L'ÉTUDE 35

CHAPITRE 6 : 37

6. ANALYSE ET SPÉCIFICATION 37

6.1. INTRODUCTION 37
6.2. L'HYPERTEXTE 37
6.3. HTML ET SON ENVIRONNEMENT 38
6.3.1. Qu'est ce que HTML? *38*
6.3.2. Dans quel contexte s'exécute HTML ? *38*
6.3.3. Le protocole d'adressage des documents - URL *39*
6.3.4. Structure du document HTML *40*
6.4. FORMULAIRE ET CGI 41
6.4.1. Le principe du formulaire *41*
6.4.2. Description des balises de formulaire *42*

6.4.3. Document HTML interactif *45*
6.5. LA PROGRAMMATION CGI 45
6.5.1. Introduction *45*
6.5.2. Emplacement des scripts cgi *46*
6.5.3. Les langages de programmation et la sécurité *47*
6.5.4. Quelques mots sur la sortie standard d'un scripts CGI *47*
6.5.5. Les variables d'environnement et les CGI *50*
6.6. APPORTS DE LA TECHNOLOGIE JAVA 54
6.6.1. Introduction *54*
6.6.2. Servlets *55*
6.6.3. A quoi servent les servlets ? *56*
6.6.4. Les avantages des servlets ? *56*
6.6.5. Les différences entre les scripts CGI et les servlets *57*
6.6.6. Mise en œuvre des servlets *57*
6.6.7. Passage des informations à une servlet par un formulaire HTML au moyen d'une méthode POST ou GET *57*
6.7. ANALYSE DES DOCUMENTS RETOURNÉS PAR LES MÉTA-MOTEURS 58
6.8. TRAITEMENTS À RÉALISER 61
6.8.1. Génération de formulaire *63*
6.8.2. Formulation de la requête *63*
6.8.3. Envoi de la requête au méta-moteurs *63*
6.8.4. Trouver Documents Répondant *63*
6.8.5. Constitution de la base des données des références *63*
6.8.6. Calcul des poids de termes dans un document *68*
6.8.7. Calcul de la similarité entre requête et les documents *69*
6.8.8. Détermination des documents pertinents et générations des nouveaux termes pour le feedback 69
6.8.9. Evaluation du client et feeback *70*

CHAPITRE 7 : **71**

7. IMPLÉMENTATION **71**

7.1. INTRODUCTION 71
7.2. CLASSE NODE 72
7.3. CLASSE OBJECTNOTFOUNDEXCEPTION 72
7.4. CLASSE LINKLIST 72
7.5. CLASSE ARBRE 73
7.6. CLASSE REQUETE 74
7.7. CLASSE ENVOIREQUETE 74
7.8. CLASSE CHERCHEREFERENCE 75
7.9. CLASSE RACINE 77
7.10. CLASSE DMRACINE 77
7.11. CLASSE DOCUMENT 78
7.12. CLASSE DOCUMENTMANAGER 79
7.13. CLASSE SIMILARITY 80
7.14. CLASSE REPONSETOCLIENT 80
7.15. CLASSE PORTER 81
7.16. CLASSE VUE_BD 81
7.17. CLASSE BD 83
7.18. CLASSE COORDINATOR 83

CHAPITRE 8 : **84**

8. CONCLUSION ET PERSPECTIVES **84**

INDEX BIBLIOGRAPHIQUE **85**

Chapitre 1 :

1. Introduction générale

Lorsque l'on effectue une recherche sur le Web, à l'aide de mots clés, on obtient généralement une liste des références très nombreuses et pas nécessairement pertinentes. Pour améliorer la qualité des références fournies, on doit faire intervenir des notions *d'information retrieval* (recherche d'information) qui est une science dont le but est la gestion de grosses quantités d'information. Les éléments de l'information peuvent être du texte, des graphiques, du son, de la vidéo.

Un utilisateur accède aux systèmes de recherche d'information , en leur soumettant une requête, le système essaie de trouver tous les documents qui "satisfont" à la requête. Les Systèmes de Recherche d'Information (SRI) ne donnent pas une réponse exacte, mais produisent une liste de documents qui semblent contenir l'information pertinente à la requête, d'où la nécessité d'avoir des systèmes de recherche d'information efficace.

En effet, quand nous cherchons de l'information, l'information que les SRI nous envoient correspond -t- elle à celle que nous souhaitons recevoir?

Les bases de données relationnelles répondent bien à cette question. Dans ces types de recherche d'information , on a une recherche d'information structurée, il suffit de connaître l'identifiant de l'information recherchée.

Par opposition à l'information structurée, nous sommes toujours confrontés à des SRI où l'information recherchée est non structurée. Ces informations non structurées n'ont pas d'identifiant, et la recherche va porter sur les termes présents dans des textes sans connaître à priori les résultats.

Dans ces systèmes de recherche d'information non structurée, différents travaux ont été et continuent à être réalisés. Les techniques *d'information retrieval* sont appliquées pour que l'information trouvée corresponde bien à l'information recherchée.

Grâce à l'*information retrieval,* les méthodes d'indexation de documents et les méthodes d'aide à la formulation des requêtes, ont été mises au point pour que l'information restituée à une requête soit celle qui est plus similaire à cette requête.

Malgré les applications de ces techniques de *l'information retrieval*, beaucoup de travaux restent à réaliser. Par exemple quand on cherche une information sur l'Internet avec des moteurs de recherche, la pertinence des documents restitués par ces derniers laisse souvent à désirer. En effet, le plus souvent , ces moteurs de recherche donnent des documents qui ne répondent pas du tout au domaine de l'information recherchée. Pour pallier à ce problème, l'intervention de l'expert du domaine devient nécessaire pour juger la qualité des références fournies par ces systèmes de recherche d'information.

Dans ce travail, nous allons exploiter les techniques de *l'information retrieval* pour évaluer la pertinence des documents restitués par des moteurs de recherche sur le Web, un aperçu théorique de *l'information retrieval* est donné, les différents moteurs de recherche sont comparés, les techniques de la programmation Internet sont énumérées et une spécification du prototype à mettre en œuvre pour l'amélioration de la qualité des références fournies par ces moteurs de recherches sur le Web sera réalisée.

Le prototype à mettre au point, non seulement, permet le suivi de l'apparition d'un nouveau document répondant à une requête donnée, mais aussi il permet d'évaluer la similarité de la requête avec les documents restitués en utilisant les méthodes statistiques de l'*information retrieval.* La pertinence des documents est toujours évaluée en collaboration avec l'expert du domaine.

Chapitre 2 :

2. Mesure de l'efficacité de recherche d'information

La mesure de l'efficacité de la recherche d'information se mesure par le *rappel* et la *précision*

Van Rijsbergen (1979), Salton (1983), Bärtschi(1984) et Girardi Gutierrez (1995) définissent ces deux termes dans ces mots :

Le *rappel (recall* en anglais) correspond à un nombre de documents restitués à une requête. C'est à dire la capacité du système de recherche d'information (SRI) de trouver des information répondant à la requête.

La *précision* est une mesure de pertinence des documents restitués à une requête. C'est à dire la capacité du SRI de fournir des information pertinentes à la requête qui lui est soumise.

Un bon système de recherche d'information doit optimiser à la fois le *rappel* et la *précision*. La figure 2.1 illustre le calcul de mesures de *rappel* et de la *précision.*

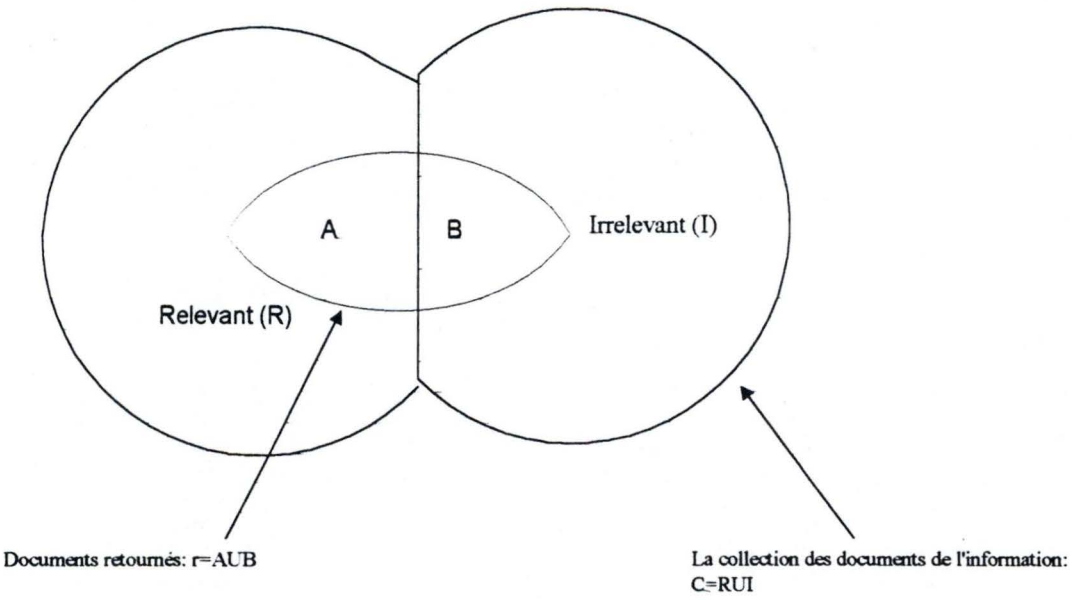


Figure 2.1:Mesure de *rappel* et de *précision*

Pour une requête donnée et pour certains critères d'évaluation de la pertinence de documents ; la collection C de documents est composée d'un ensemble R de documents jugés pertinents (*Relevant* sur la figure 2.1) à une requête et de l'ensemble I (*Irrelevant* sur la figure 2.1) des documents considérés non pertinents.

La liste r des documents restitués à une requête est composée de deux ensembles A et B. L'ensemble A contient des documents jugés pertinents et l'ensemble B concerne les documents non pertinents ($A \subseteq R$, $B \subseteq I$).

Le *rappel* est la proportion des documents pertinents restitués tandis que la *précision* est la proportion des documents restitués par le SRI que l'on juge pertinents à la requête.

$$Rappel = \frac{|A|}{|R|} = \frac{Nombre\ de\ documents\ pertinents\ restitués}{Nombre_total\ de\ documents\ pertinents\ dans\ la\ collection} \qquad (2.1)$$

$$\text{Pr}\acute{e}cision = \frac{|A|}{|r|} = \frac{Nombre\ de\ documents\ pertinents\ restitués}{Nombre\ total\ de\ documents\ restitués} \qquad (2.2)$$

La figure 2.2 montre la relation qui existe entre le *rappel* et la *précision* dans les systèmes de recherche d'information (SRI). Dans ces systèmes le rappel peut augmenter tant que le nombre de documents restitués augmente, parce que les nouveaux documents peuvent être pertinents à une requête. Ce pendant, la précision ,elle, semble diminuer. Généralement, les mécanismes qui améliorent le *rappel* affectent d'une façon défavorable la *précision* et vice versa.

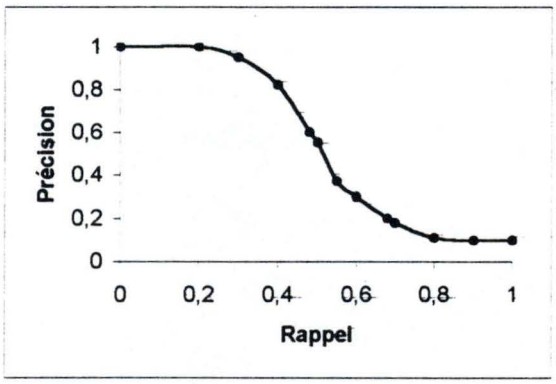


Figure 2.2 : Relation entre *Rappel* et *Précision*

Chapitre 3 :

3. Les facteurs affectant l'efficacité d'une recherche d'information

3.1. Introduction

Beaucoup de facteurs affectent l'efficacité des Systèmes de Recherche d'Information :

- des méthodes d'indexation ;
- les mécanismes utilisés pour le « *matching* » et l'analyse de la similarité entre des requêtes et des documents;
- la stratégie de recherche.

3.2. Les méthodes d'indexation

Les méthodes d'indexation sont à la base des facteurs qui influencent l'efficacité de recherche d'information.

Lors de l'indexation, la première tâche et qui n'est pas facile, consiste aux opérations d'analyse du document afin de trouver des termes capables de représenter le contenu du document. En principe, le processus d'analyse de document est superflu si la collection de documents est assez petit pour permettre de parcourir le texte complet de tous les documents chaque fois que la requête est formulée. Mais dans la pratique, cette solution consomme trop de temps. C'est pourquoi, d'habitude, on va caractériser chaque document de la collection par une petite description ou profil, qui peut être utilisé pour l'obtention d'accès au document chaque fois que ce dernier est demandé. Classiquement, dans les bibliothèques, ces opérations d'analyse de document, sont connues sous le vocable de catalogue, indexation, classification, et résumé.

Le profil de document obtenu par ces opérations d'analyse porte le nom de *représentant de document*.

Le *représentant de document* en *information retrieval* est en général un vecteur de mots clés, éventuellement, munis d'une pondération. Le profil de document peut

aussi être muni d'informations bibliographiques, telles que les noms de l'auteur, l'éditeur, ou la date de parution.

L'indexation doit viser trois objectifs comme le préconise Salton (1983) et repris par D'Haeyere(1995, p.26)

- *permettre à un utilisateur de localiser des documents selon le sujet traité;*
- *relier des documents entre eux et les relier au sujet traité ou à un sujet proche;*
- *permettre d'établir la pertinence d'un document particulier par rapport à une demande précise.*

Souvent, il est nécessaire de prendre en compte les deux caractéristiques de l'index, à savoir l'*exhaustivité* de l'index et la *spécificité* de l'index .

L' *exhaustivité* de l'index fait référence à la façon dont tous les concepts et les notions traités dans le document sont repris dans l'index décrivant ce document. L' *exhaustivité* de l'index accroît le *rappel*, car la proportion des documents pertinents peuvent être restitués par le système de recherche d'information.

La *spécificité* de l'index, elle, fait référence au niveau générique des termes utilisés dans l'index pour caractériser le contenu du document. Plus les termes utilisés dans l'index sont précis et spécifiques, moins le nombre de documents restitués à l'utilisateur suite à une requête est élevé.

Selon plusieurs auteurs (Van Rijsbergen,1979; Salton,1983; Girardi Gutierrez,1995; D'Haeyere ,1995) l' *exhaustivité* élevée de l'index conduit à un *rappel* élevé et une faible *précision* et inversement, l'*exhaustivité* faible de l'index conduit à une *précision* élevée. Quant à la *spécificité*, on constate que la *spécificité* élevée de l'index conduit à une *précision* élevée et un *rappel* faible et vice versa.

Il semble donc qu'il y a un niveau optimal à trouver pour *l'exhaustivité* de l'index et sa *spécificité* lors des tâches d'indexation.

L'indexation d'un document peut se faire de deux façons:

- indexation manuelle;
- indexation automatique.

3.2.1. Indexation manuelle

C'est un travail effectué par un expert du domaine traité dans le document. L'expert doit assigner des mots clés à un document. Le choix du vocabulaire à utiliser est, soit libre ou contrôlé. Quand il s'agit du vocabulaire contrôlé, l'expert doit utiliser les mots clés se trouvant dans une liste des mots clés établie au préalable.

3.2.2. Indexation automatique

Le représentant d'un document, ou l'index, est établi d'une façon automatique par un programme.

Deux types des méthodes d'indexation automatiques existent en *information retrieval* :

- les méthodes statistiques;
- les méthodes linguistiques.

Dans ce travail , nous allons utiliser les méthodes d'indexation automatiques basées sur la statistique.

Méthodes statistiques d'indexation automatique

Considérations générales

La tâche d'indexation consiste d'abord à assigner les termes ou les concepts à chaque document stocké capable de représenter le contenu de ce document et ensuite, la tâche d'indexation va être d'assigner le poids (ou une valeur) à chaque terme reflétant l'importance présumé de ce terme pour l'identification du contenu du document. Evidemment, la première place où on peut trouver l'identification du contenu est le texte des documents eux-mêmes, les titres de ces documents ou leurs résumés.

L'indexation automatique en *information retrieval*, se base sur les hypothèses que la fréquence d'un mot dans un document texte est représentative de l'importance de ce mot et du concept de ce mot dans le document.

Particulièrement, si tous les mots apparaissent, aléatoirement, dans tous les documents de la collection avec des fréquences égales, il serait impossible de distinguer entre eux les critères de leur utilisation comme termes d'index. Mais il a été observé que dans les textes de langage naturel, les mots apparaissent d'une façon irrégulière. Par conséquent, les classes de mots peuvent être distinguables par leur fréquence d'occurrences.

On a pu constaté que ni les termes très fréquents ni les termes moins fréquents ne sont pas de bons identificateurs du contenu de document.

Par ailleurs, l'élimination de tous les termes de fréquence élevée peut produire une perte de *rappel* , parce que l'utilisation de termes généraux, c'est à dire dont la fréquence est très élevée est effective pour la restitution d'un grand nombre de documents. A l'inverse, l'élimination des termes moins fréquents peut provoquer la perte de *précision*.

Un autre problème est de choisir les seuils appropriés (borne inférieure et borne supérieure) pour trouver les termes de fréquence intermédiaire utiles. Finalement, la question est de savoir si on va utiliser la fréquence absolue de chaque terme dans le document (*FREQik* ou la fréquence totale du terme dans toute la collection de document (*TOTFREQk*) pour l'identification du contenu de documents. La raison en est que les bons termes d'index doivent remplir deux fonctions: un meilleur *rappel* et une meilleur *précision.*

De tous ces problèmes, différentes formules d'attribution de poids à un terme sont proposées, avec leur défauts et leurs avantages, dans ce travail, la formule normalisée est retenue. Cette formule est sous la forme suivante :

$$w_{ik} = \frac{tf_{ik} * \log\left(\frac{N}{n_k}\right)}{\sqrt{\sum_{j=1}^{t} (tf_{ij})^2 * \left(\log\left(\frac{N}{n_j}\right)\right)^2}} \qquad (3.1)$$

Où w_{ik} est le poids du terme T_k dans le document D_i;

tf_{ik} est la fréquence d'occurrence du terme T_k dans le document D_i;

N est le nombre de documents dans toute la collection;

n_{ik} est le nombre de documents comprenant le terme T_k;

t est le nombre de termes distincts dans la collection de documents.

Pour créer l'index les étapes suivantes sont suivies :

- l'élimination des mots fréquents;
- générer les racines communes ;
- agrégation des fréquences d'occurrence pour les mots ayant une racine commune.

Il a été démontré que (Salton, 1983; Girardi Gutierrez, 1995; D'haeyere, 1995) :

- *l'utilisation de la pondération par l'inverse de la fréquence d'occurrence dans le document est plus efficace que l'utilisation d'une autre pondération;*
- *une technique d'élimination des suffixes pour les termes apparaissant rarement augmente le taux de rappel du système;*
- *la combinaison des termes de fréquence élevée en phrases augmente la précision du système.*

Association automatique des termes et utilisation du contexte

Lors de l'indexation, que ce soit l'indexation manuelle ou automatique, la considération du contexte semble améliorer l'efficacité des systèmes de recherche d'information.

Dans l'indexation automatique, l'association des termes fait intervenir l'utilisation des *thesauri*. Ces *thesauri* sont construits manuellement ou d'une façon automatique par un programme (Van Rijsbergen, 1979; Salton, 1983).

On distingue deux types de *thesauri* :

- le premier type concerne des thesauri qui relient les mots interchangeables, c'est à dire que ces termes sont repris dans des classes équivalentes. Un terme A est considéré comme équivalent, donc interchangeable, à un terme B, si l'utilisateur accepte que le système restitue des documents contenant le terme B alors que la requête portait sur le terme A. Dans des classes équivalentes, chaque mot peut être choisi pour représenter chaque classe et une liste de ces mots peut être utilisée pour former un vocabulaire contrôlé (Van Rijsbergen, 1979);
- le deuxième type de *thesauri* est constitué par des thesauri qui utilisent des relations sémantiques existant entre les mots. Ces *thesauri* reprennent des relation hiérarchiques entre les termes de l'index .

Deux approches citées (Van Rijsbergen, 1979; D'haeyere, 1995) pour l'utilisation des *thesauri* sont :

- remplacer chaque mot clé dans le représentant de document (ou de requête) par le nom de la classe (qui est en fait le terme générique de cette classe) dans laquelle il apparaît;
- remplacer chaque mot clé par tous les mots clés apparaissant dans la classe à laquelle il appartient.

3.3. Fonctions de correspondance et mesures de la similarité

L'analyse de la similarité permet de faire une classification des documents correspondant à une requête, et la pertinence est évaluée en fonction de l'information recherchée.

Les documents les plus similaires à la requête vont être placés les premiers à la liste des documents restitués à la requête permettant ainsi à l'utilisateur de les analyser les premiers. Par conséquent, la qualité des mesures de similarité est un facteur important pour le contrôle de l'efficacité de recherche d'information. Les mesures de similarité de bonne qualité vont assurer une bonne *précision* en plaçant les documents les plus similaires à la requête au premier rang du classement, elles peuvent aussi assurer le *rappel* par la considération de correspondance partielle des structures internes (*sémantiques*) comparées (Girardi Gutierrez, 1995).

Alors comment évaluer la similarité ?

Pour mesurer la similarité d'un document avec une requête, ou pour mesurer la similarité de deux document, on va se baser sur les modèles de représentation des documents (ou des requêtes) par des vecteurs de mots clés.

Si on considère deux documents i et j avec leurs représentants respectifs D_i et D_j et si on admet que $TERM_{ik}$ représente le poids du terme k assigné au document i. Dans les systèmes dits *binaires*, la valeur de terme $TERM_{ik}$ est 1 si le terme k existe dans le document i, et 0 si ce terme est absent. Sinon, dans d'autres systèmes, les poids $TERM_{ik}$ varient de zéro à une valeur maximale déterminée dans la collection de recherche, mais pour les systèmes normalisés, $TERM_{ik}$ est compris entre 0 et 1.

Si t est le nombre de termes pour le *représentant de document*, alors on a :

$$D_i = (TERM_{i1}, TERM_{i2}, TERM_{i3} \ldots TERM_{it}) \qquad (3.3)$$

$$D_j = (TERM_{j1}, TERM_{j2}, TERM_{j3} \ldots TERM_{jt}) \qquad (3.4)$$

Di et *Dj* étant des *représentants* de deux documents (ou requête) respectifs *i* et *j*.

Pour évaluer la similarité entre les deux documents , différentes formules sont proposées : Coefficient de *Dice*, Coefficient de *Jaccard*, Coefficient *cosinus*, mesure de recouvrement (*overlap measure*) et mesure asymétrique (*asymmetric measure*). Toutes ces formules sont illustrées ci dessous.

Coefficient de Dice :

$$SIM(D_i, D_j) = \frac{2\left(\sum_{k=1}^{t}(TERM_{ik} * TERM_{jk})\right)}{\sum_{k=1}^{t} TERM_{ik} + \sum_{k=1}^{t} TERM_{jk}} \qquad (3.5)$$

Coefficient de *Jaccard*:

$$SIM(D_i, D_j) = \frac{\sum_{k=1}^{t}(TERM_{ik} * TERM_{jk})}{\sum_{k=1}^{t} TERM_{ik} + \sum_{k=1}^{t} TERM_{jk} - \sum_{k=1}^{t}(TERM_{ik} * TERM_{jk})} \qquad (3.6)$$

Coefficient *Cosinus* :

$$SIM(D_i, D_j) = \frac{\sum_{k=1}^{t} (TERM_{ik} * TERM_{jk})}{\sqrt{\sum_{k=1}^{t} (TERM_{ik})^2 * \sum_{k=1}^{t} (TERM_{jk})^2}} \qquad (3.7)$$

Overlap measure:

$$SIM(D_i, D_j) = \frac{\sum_{k=1}^{t} (TERM_{ik} * TERM_{jk})}{\min\left(\sum_{k=1}^{t} TERM_{ik}, \sum_{k=1}^{t} TERM_{jk}\right)} \qquad (3.8)$$

Asymmetric measure:

$$SIM(D_i, D_j) = \frac{\sum_{k=1}^{t} \min(TERM_{ik}, TERM_{jk})}{\sum_{k=1}^{t} TERM_{ik}} \qquad (3.9)$$

Le coefficient de *Jaccard* et le coefficient *cosinus* ont des caractéristiques semblables , ils rangent la valeur de la similarité entre 0 et 1 pour les vecteurs des éléments non négatifs. Ces deux mesures sont considérées comme étant plus faciles à calculer et elles apparaissent plus efficaces dans les systèmes de recherche d'information.

Concernant le coefficient *cosinus* Goffinet *et al*(1996) ont pu montré son efficacité lors de la création des documents hypertextes.

3.4. Les stratégies de recherche d'information

3.4.1. Introduction

Toutes les stratégies de recherche d'information sont basées sur les comparaisons qui s'effectuent entre la requête et les documents de la collection. De temps en temps, ces comparaisons sont accomplies indirectement quand la requête est comparée avec le *cluster* (ou plus précisément avec les *représentants* des *clusters*).

Les différences entre les types de stratégies peuvent être comprises en regardant les langages de requête, c'est à dire le langage dans lequel l'information recherchée est exprimée. La nature du langage de requête souvent dicte la nature de la stratégie de recherche. Par exemple, un langage de requête permettant la déclaration d'une requête en faisant la combinaison logique des mots clés dicte normalement la stratégie de recherche booléenne.

3.4.2. Recherche booléenne

La stratégie de recherche booléenne est une stratégie qui réalise les résultats de recherche par comparaisons logiques de la requête avec les documents. Cette stratégie restitue tous les documents qui sont "vrais" pour la requête.

Cette formulation de requête a de sens si les requêtes sont exprimées dans les termes utilisés comme mots clés lors de l'indexation et combinées avec les opérateurs logiques(OR, AND, NOT,...).

La stratégie de recherche booléenne est facilement implémentée dans les systèmes d'information utilisant des structures de fichier indexé inversé où à chaque terme de l'index est associé une liste de documents reprenant ce terme. Les opérations ensemblistes (union, intersection, différence) effectuées sur ces types de fichiers permettent d'obtenir des résultats.

Les systèmes de recherche d'information avancés ont introduits des opérateurs permettant d'insister sur certains termes ou sur la cooccurrence de ces termes. C.LELOUP (1997) parle de la recherche textuelle.

3.4.3. Recherche textuelle

Troncature et masque

Les opérateurs de recherche disponibles dépendent du type d'index et, évidemment, de ce que le Système de Recherche d'Information (SRI) sait faire. Les opérateurs accessibles sont essentiellement de deux types :

- la *troncature* : c'est un caractère spécial, qui remplace une chaîne de caractères quelconque;
- le *masque* : c'est un caractère spécial qui remplace un ou aucun caractère.

On prend le plus souvent la convention d'utiliser le caractère ? pour la troncature et le caractère # pour le masque.

La troncature peut être gauche, milieu, ou droite. Par exemple (LELOUP, 1997), la question sur *informati*? ramènera les documents comprenant les termes *information, informations, informatique, informatiques, informaticien...* soit tout ce qui commence par *informati.* Il s'agit là d'une troncature droite.

La troncature gauche par exemple avec la requête *?informatique*, le système de recherche va fournir tous les documents comprenant les termes *micro-informatique, informatique, téléinformatique...*

La troncature du milieu par exemple avec le mot *anticonstitutionnellement;* avec la requête *anticonstitution?ment,* on obtient tous les documents qui comprennent ce mot même s'il manque quelques *n* ou *l.*

Quant au masque, la question sur *micro#informatique* va ramener des documents comprenant *micro-informatique* ou *microinformatique.*

Opérateurs de proximité

Pour les SRI supportant les requêtes en texte intégral, on peut bénéficier des opérateurs de proximité. Ces opérateurs de proximité permettent de spécifier les positions des mots cherchés.

On distingue classiquement trois opérateurs de proximité (C. LELOUP, 1997) :

- l'adjacence;
- la distance;
- la présence dans une même portion de texte.

L'adjacence est l'opérateur qui permet de chercher deux termes adjacents, c'est à dire situés l'un à côté de l'autre, les mots vides (*stopswords)* n'étant pas pris en compte. L'ordre de présentation peut être spécifiés ou être indifférent. Par exemple (C. LELOUP, 1997) si on cherche une information sur les *vins rouges*, on va formuler la requête de la façon suivante : *vin*? adj *rouge* ("adj" pour signifier l'opérateur adjacence).

La *distance* est un opérateur qui permet d'imposer une contrainte de distance maximale entre les mots recherchés, les mots vides étant ignorés. L'ordre d'apparition des mots peut être spécifié ou non. Par exemple (C. LELOUP, 1997) en utilisant l'abrégé dist pour désigner cet opérateur :

- si on s'intéresse aux interactions entre *l'informatique* et la *télécommunication*, on peut poser la question de la manière suivante: *informati*?dist3 *télécom*?, sans préciser l'ordre;
- avec la requête *informati*?dist+3 *télécom*? Le caractère + signifie qu'il faut respecter l'ordre.

La contrainte d'appartenir à une même portion de texte est, en général, limité à la phrase ou au paragraphe. On peut également préciser l'ordre. Par exemple (C. LELOUP, 1997) pour la recherche sur *l'informatique* et la *télécommunication,* on peut formuler la requête de la manière suivante : *informati*?phrase *télécom*?, ou pour élargir la recherche par *informati*?para *télécom*?, sans préciser l'ordre.

3.4.4. Recherche par les fonctions de correspondance

La plupart des stratégies sophistiquées sont implémentées par les fonctions de correspondances (*matching functions*). Ces fonctions mesurent la similarité entre la requête et les documents (ou le *représentant* de *clusters*).

Souvent, on différencie les mesures de similarité et les fonctions de correspondance, du fait que ces dernières mesurent la similarité entre la requête et les documents, tandis que les mesures de similarité s'effectuent entre des objets de même type (par exemple entre deux documents). Mais mathématiquement, ces deux fonctions ont les mêmes propriétés, il n'y a que l'interprétation qui les différencie.

La fonction de correspondance la plus utilisée est celle de la *corrélation cosinus* (voir point 3.3) qui stipule que le document D et la requête Q sont représentés par les vecteurs en t dimensions :

$$Q = (q_1, q_2, q_3, \ldots, q_t) \qquad (3.10)$$

$$D = (d_1, d_2, d_3, \ldots, d_t) \qquad (3.11)$$

La similarité entre la requête Q et le document D est calculée de la façon suivante:

$$SIM(D,Q) = \frac{\sum_{i=1}^{t}(d_i * q_i)}{\sqrt{\sum_{i=1}^{t}(d_i)^2 * \sum_{k=1}^{t}(q_i)^2}} = \cos inus(\theta) \qquad (3.12)$$

d_i = le poids du terme i dans le document D

q_i = le poids du terme i dans la requête Q

3.4.5. Recherche séquentielle (*serial search*)

Bien que la stratégie de recherche séquentielle soit connue d'être lente, néanmoins, elle reste utilisée dans la plupart des Systèmes de Recherche d'Information.

Supposer qu'il y ait N documents D_i dans une collection, la recherche séquentielle consiste à balayer cette collection et comparer chaque document un à un à la requête.

Deux approches sont appliquées dans cette stratégie :

(1) une fonction de correspondance détermine un seuil de similarité convenable,

les documents dont la similarité avec la requête est supérieure au seuil sont jugés pertinents. Si T est le seuil considéré, l'ensemble B des documents restitués par le système est :

$$B = \{Di \,/ Sim(Di,Q) > T\} \qquad (3.13)$$

(2) Une autre approche est celle où les documents sont rangés en ordre croissant

selon la valeur de la fonction de correspondance. Une borne R est choisie comme un seuil et tous les documents dont le classement est inférieur à ce seuil R sont restitués à l'utilisateur. L'ensemble B des documents restitués est :

$$B = \{Di \,/ r(i) < R\} \qquad (3.14)$$

où $r(i)$ est le classement assigné au document Di.

De toutes ces deux approches , la difficulté principale est l'évaluation du seuil ou niveau r de classement à partir duquel les documents vont être sélectionnés. C'est toujours arbitraire et souvent par essais et erreur que ces évaluations sont effectuées.

3.4.6. Recherche dans les *clusters*

Dans cette stratégie de recherche, la requête est comparée aux représentants des *clusters*, si la valeur de la fonction de correspondance est élevé, alors les documents constituant le *cluster* sont pertinent à la requête et ils sont restitués ou bien on peut continuer la recherche uniquement à l'intérieur des *clusters* intéressants (Van Rijsbergen, 1979, D'haeyere, 1995).

3.4.7. Stratégie de *relevance feedback*

Introduction

Il est bien connu que la formulation de requête initiale à soumettre aux Systèmes de Recherche d'Information (SRI) n'est pas transparente pour la plupart des utilisateurs de ces systèmes.

L'efficacité des SRI est influencée par la proximité de la requête avec l'information recherchée . La figure 3.1 illustre deux situations qui peuvent arriver lors de la spécification d'une requête.

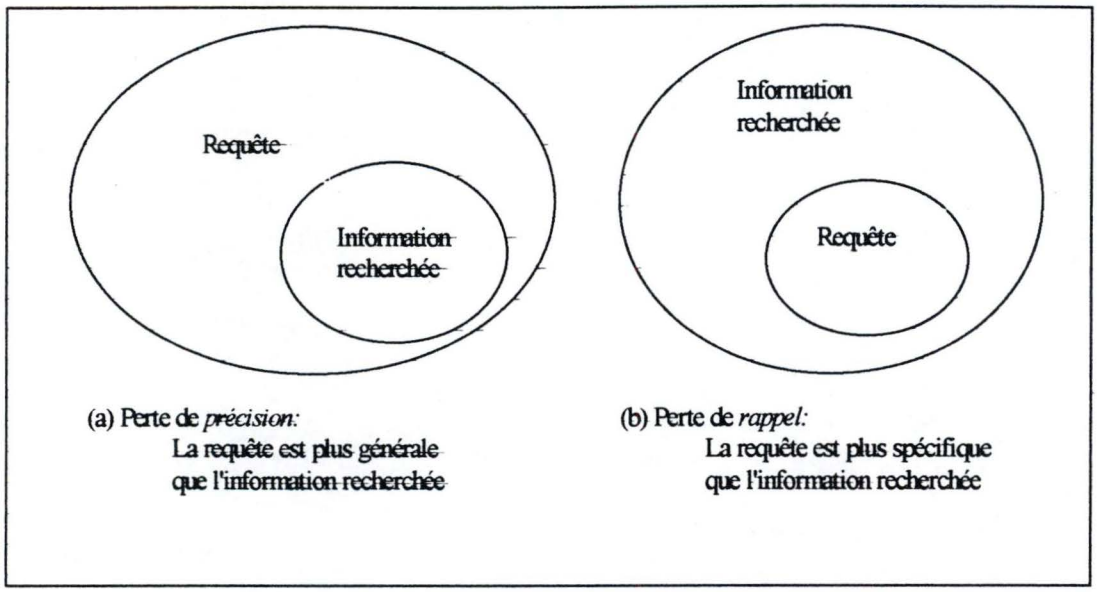


Figure 3.1 : L'influence de la formulation de requête sur l'efficacité de recherche d'information (GIRARDI GUTIERREZ, 1995)

Dans la situation (a) la requête est plus générale que l'information recherchée. Par conséquent, beaucoup de documents restitués peuvent être non pertinents et il y a une perte de *précision*. Cette situation est très fréquente parce que le plus souvent l'utilisateur suppose que le système opère au niveau plus large mais non pas au niveau spécifique .

La situation (b) est moins fréquente, la requête est plus spécifique que l'information recherchée et par conséquent, on peut perdre le *rappel*, parce que beaucoup de documents pertinents ne peuvent pas être restitués.

Ces deux pertes, de *précision* et de *rappel,* apparaissent dans les SRI basés sur l'utilisation des mots clés simples, quand les utilisateurs n'arrivent pas à trouver des mots clés adéquats, ou quand ils n'arrivent pas à faire des combinaisons booléennes de ces mots clés d'une façon convenable ou quand ils n'arrivent pas à trouver des mots synonymes ou très proches de ces mots clés afin d'exprimer d'une façon adéquate l'information qu'ils cherchent.

Autrement , les utilisateurs sont aidés par les interfaces en langage naturelle qui leur permettent d'exprimer leur requête.

Les Systèmes de Recherche d'Information (SRI) abordent le problème de formulation de requête par un processus appelé *relevance feedback*. Ce processus supporte une construction incrémentale d'une requête par une évaluation de documents restitués par le système. C. LELOUP (1997) parle des "aides à la recherche", car les outils, en fournissant de nouveaux termes à l'utilisateur aident ce dernier lors de l'affinement ou élargissement de sa recherche à l'issue de l'obtention des premiers résultats. Dans cette aide on peut aussi utiliser des thésauri.

La *relevance feedback* est une technique qui permet à l'utilisateur d'améliorer sa requête initiale afin d'obtenir des meilleurs résultats de sa recherche d'information dans les SRI.

Dans les opérations de *relevance feedback*, on distingue trois méthodes :

- les méthodes dites « *Vector Processing Methods* » ;
- les méthodes dites « *Probabilistics feedback Methods* »
- les méthodes basés sur les réseaux de neurones dites « *Neural relevance feedback* »

Vector processing Methods

Selon Salton (1988), dans les méthodes de *Vector Processing Methods* , une requête Q_0 est représentée par un vecteur qui reprend le poids de chaque terme utilisé dans la requête . Q_0 est de la forme suivante :

$$Q_0 = (q_1, q_2, q_3, \ldots q_i) \qquad (3.15)$$

Où q_i représente le poids du terme i dans la requête Q_0.

Dans les systèmes de recherche d'information, le document D est présenté par un vecteur de poids de termes caractérisant ce document .

$$D = (d_1, d_2, d_3, d_i) \qquad (3.16)$$

où d_i représente le poids du terme i dans le document D.

La similarité entre la requête Q_0 et le document D va être calculée selon la formule suivante :

$$Sim(D,Q_0) = \sum_{i=1}^{t} d_i * q_i \qquad (3.17)$$

t = nombre de termes dans le représentant de document et de requête

d_i = le poids du terme i dans le *représentant* du document D.

q_i = le poids du terme i dans le *représentant* de la requête Q_0.

On constate que pour calculer la similarité de la requête et le document, on effectue le produit scalaire du vecteur représentant le document et le vecteur représentant la requête. Il est connu que dans un environnement utilisant le produit scalaire pour calculer la similarité entre le vecteur de requête et le vecteur de document, la bonne requête conduit à une recherche de n documents particuliers (qui sont

pertinents) à partir de N documents de toute la collection (Salton, 1988). Cette bonne requête (Q_{opt})est de la forme ci dessous :

$$Q_{opt} = \frac{1}{n} \sum_{i \in pertinent} \frac{D_i}{|D_i|} - \frac{1}{N-n} \sum_{i \in non\ pertinent} \frac{D_i}{|D_i|} \qquad (3.18)$$

En pratique, cette formule est inutilisable, dans la mesure où, dès le départ, lors de la formulation de la requête initiale Q_0 , on ne connaît pas encore l'ensemble des documents pertinents. Cette requête initiale Q_0 va être améliorée en ajoutant ou en soustrayant de termes des termes pris dans des documents jugés pertinents ou non pertinents, on obtient ainsi l'équation suivante :

$$Q_1 = Q_0 + \frac{1}{n_1} \sum_{i \in pertinent} \frac{D_i}{|D_i|} - \frac{1}{n_2} \sum_{i \in non\ pertinent} \frac{D_i}{|D_i|} \qquad (3.19)$$

Dans cette formule, le $n_1 et$ le n_2 représentent le nombre de documents jugés pertinents et l non pertinents, respectivement.

Q_0 et Q_1 représentent, respectivement, la requête initiale et la requête de la 1ère itération.

Généralement, la formulation de la requête prend cette forme :

$$Q_{i+1} = \alpha Q_i + \beta \frac{1}{n_1} \sum_{i \in pertinent} \frac{D_i}{|D_i|} - \gamma \frac{1}{n_2} \sum_{i \in non\ pertinent} \frac{D_i}{|D_i|} \qquad (3.20)$$

α, β, γ étant des constantes.

Et comme le plus souvent ces formules sont utilisées après la normalisation, le poids de chaque terme va être entre 0 et 1.

Probabilistic Feedback Methods

Une méthode alternative de *relevance feedback est* basée sur les modèles de recherche prababiliste (Salton, 1988 ; Crestani, 1995 ;).

Dans ces modèles, la similarité entre la requête Q et le document D se mesure dela façon suivante :

$$Sim(D,Q) = \sum_{i=1}^{t} d_i * \log \frac{p_i(1-u_i)}{u_i(1-p_i)} + cons\tan te \qquad (3.21)$$

t= le nombre de termes dans le représentant de documents

p_i et u_i représentent, respectivement, la probabilité que le $i^{ème}$ terme aie la valeur 1 dans les documents pertinents et non pertinents.

$$p_i = pr(x_i = 1 / pertinent) \qquad (3.22)$$

$$u_i = pr(x_i = 1 / non\ pertinent) \qquad (3.23)$$

Avec les ajustements, on arrive à calculer p_i et u_i et on obtient :

$$p_i = pr(x_i = 1 / pertinent) = \frac{r_i + \frac{n_i}{N}}{R+1} \qquad (3.24)$$

$$u_i = pr(x_i = 1 / non\ pertinent) = \frac{n_i - r_i + \frac{n_i}{N}}{N-R+1} \qquad (3.25)$$

Crestani (1995) pour améliorer la requête, les termes à ajouter à la requête

initiale sont choisis dans les premiers m termes d'une liste où tous les termes présents dans les documents pertinents sont classés selon une fonction de pondération suivante:

$$w_i = \log \frac{r_i(N - n_i - R + r_i)}{(R - r_i)(n_i - r_i)} \tag{3.26}$$

Dans cette formule, on a :

w_i = le poids du terme i dans le document;

N= nombre de tous les documents présents dans la collection;

n_i = nombre de document avec au moins une occurrence du terme i;

R = nombre de documents jugés pertinents à une requête donnée;

r_i = le nombre de documents pertinents à une requête avec le terme i;

Comme on le constate, dans cette formule, et comme l'affirme aussi Salton (1988), l'ensemble des documents pertinents n'est pas utilisé directement pour l'ajustement de la requête dans l'environnement probabiliste, comme c'est le cas dans les modèles de vecteur. Cependant, la distribution du terme dans les documents pertinents est utilisée indirectement pour déterminer le poids de ce terme. Cette participation indirecte fait que les méthodes *relevance feedback* probabilistes ne sont pas effectives par rapport aux méthodes conventionnelles de vecteurs (Salton, 1988).

3.4.7.1. Neural relevance feedback

Récemment, les techniques de réseaux de neurone ont été introduites dans les Systemes de recherche d'information. Dans ce modèle de *Neural relevance feedback* la requête initale va être améliorée grâce au simulateur de réseaux de neurone, qui propose des termes similaires pour améliorer la requête (Crestani,1995).

Chapitre 4 :

4. Moteurs de recherche, méta-moteurs et WWW

4.1. Introduction

Les facilités de l'hypertexte mettent la puissance des *moteurs de recherche* et des *méta-moteurs* de recherche à la portée de tout le monde. Ces moteurs et méta-moteurs de recherche constituent les passerelles accédant aux immenses documents et aux bases de données disponibles sur le Web.

4.2. Définitions

Un *moteur de recherche* est une application spécialement créée et optimisée pour des recherches dans des bases de données. Avec un moteur de cherche, on soumet les mots clés à une unique base de données d'une page Web de ce moteur de recherche et comme réponse, on obtient l'affichage des documents correspondant à la requête.

La base de données d'un moteur de recherche est mise à jour par des applications "*robots*" qui tournent 24/24 heures sur Internet pour repérer des nouveaux documents qui apparaissent.

Tous ces outils (C. LELOUP, 1997) sont exclusivement spécialisés sur la recherche par le contenu d'information disponible sur le réseau Internet. Ces moteurs peuvent aussi être utilisés pour la récolte des information disponibles sur les réseaux des entreprises (Intranet). Ils ne gèrent pas des informations en propre, mais les indexent et permettent d'y accéder. Ils réactualisent des index en fonction de leurs évolutions. En d'autres termes, ils laissent l'information à sa place et ne la dupliquent pas. Ils se positionnent comme des outils de valorisation de l'accès à l'information, tant en matière de recherche que de filtrage ou de sélection de l'information disponible.

Un *méta-moteur* est aussi une application de recherche dans des bases de données mais avec un méta-moteur, on soumet des mots clés dans une "boîte" de

recherche et la requête est simultanément envoyée à plusieurs moteurs de recherche et ces moteurs de recherche vont utiliser leurs propres bases de données pour retourner des documents répondant à la requête.

Les méta-moteurs ne possèdent donc pas leur propres bases de données, ils utilisent des programmes de recherche et les bases de données de chaque moteur de recherche interrogé, et comme résultats, on obtient le plus souvent, mais pas toujours, des résultats compilés de ces moteurs de recherche.

Les méta-moteurs agissent comme des agents intelligents intermédiaires pour passer la recherche à chaque moteur et rassembler des réponses de chaque moteur de recherche, ainsi les méta-moteurs donnent des résultats unifiés provenant de plusieurs *moteurs de recherche*.

4.3. Comparaisons des méta-moteurs

4.3.1. Introduction

Partant de l'idée qu'avec les *méta-moteurs*, on sait obtenir des résultats de plusieurs moteurs de recherche simultanément, c'est pourquoi, nous avons voulu comparer les *méta-moteurs* qui existent, du point de vue de leur :

- élimination de la duplication des documents;
- logique de recherche (OR, AND, NOT,...);
- moteurs de recherche interrogés;
- logique de classement des documents retournés;
- relevance (c'est l'objet de l'expert du domaine) .

Ces comparaisons vont nous permettre comment utiliser ces méta-moteurs pour notre travail d'amélioration de la qualité des références fournies par les moteurs de recherche.

4.3.2. Comparaisons

Le tableau 4.1 montre des *méta-moteurs* avec les moteurs de recherche interrogés, la façon dont les résultats sont retournés, et la logique de recherche qu'ils utilisent pour restituer des données.

Tableau 4.1: Liste des méta-moteurs et leur façon de restituer des documents

Métamoteurs	**Moteurs utilisés**	**Combinaison des résultats**	**Logique de recherche (OR , AND, ...)**
Search.com	Infoseek, snap,altavista, Deajanews, Excite, Lycos, Yahoo	Non	Selon moteur sélectionné
All-one-search.com	Liste de moteurs et métamoteurs (Annexe I)	Non	Selon le moteur ou métamoteur sélectionné
Research-It!	Recherche d'adresses des personnes		
Savvysearch.com	Lycos, NationalDirectory, WebCrawler, Altavista, Google, Thunderstone, Infoseek, Directe Hit, HotBot, Excite, Galaxy	Oui (combinaison de 5 moteurs à la fois)	AND Phrase
Eureka!	Liste de moteur et métamoteurs	Non	Selon moteur ou métamoteur sélectionné
All4one	HotBot, Excite, Lycos, Altavista	Non Résultats de 4 sites affichés en même temps	Selon le moteur
Metacrawler	Altavista, Excite, Infoseek, LookSmart, Lycos, The Mining co., Thunderstone, Webcrawler, Yahoo	Oui	Option: Any (=OR); All (=AND) Phrase
Debriefing	Altavista, Yahoo, Infoseek, Excite, Webcrawler, Lycos, HotBot,	Oui	AND, (OR si recherche avancée) mais tantôt OR tantôt AND avec des variations très élevée
AskJeeves	Excite, Infoseek, Altavista, Lycos, Webcrawler	Non	Selon le moteur sélectionné
Dogpile	Yahoo, Thunderstone, GoTo.com, Lycos's Top 5%, Mining Co., Excite, WebCrawler, Magellan, What U Seek, Lycos, Infoseek, Excite Web Search, AltaVista	Non	Selon le moteur sélectionné
Cyber411	Altavista, Dejanews, Excite, galaxy, Goto, HotBot, LookSmart, Lycos, Magellan, PlanetSearch, Search.com, Snap, Thunderstone, WebCrawler, What-U-Seek, Yahoo	Oui	AND (OR et NOT, ... si recherche avancée)
Méta-index of web search engines	Infoseek, Net Search, Excite, Yahoo, Global Network, Academy Méta-Library, Aliweb	Non	Selon le moteur sélectionné
CUSI	Listes des moteurs et des métamoteurs	Non	Selon le moteur ou métamoteurs sélectionné
Internet Sleuth	Infoseek, Altavista, Webcrawler, Yahoo	Non	Selon le moteur sélectionné
World Searcher	Yahoo, Altavista, Lycos, infossek, Webcrawler	Non	Selon le moteur sélectionné

Tableau 4.1: Suite

Métamoteurs	**Moteurs utilisés**	**Combinaison des résultats**	**Logique de recherche (OR , AND, ...)**
Highway 61	Webcrawler, Excite, Yahoo, infossek, Lycos	Oui Avec Option : - LOTS (35 à 75 résultats sont retournés) - BURY ME (60 à 125 documents sont restitués)	Option : OR et AND
DevSEARCH	Liste des moteurs (spécialisés pour les développeurs l'Internet)	Non	Selon le moteur sélectionné
HuskySearch	Métamoteur Metacrawler	Oui Avec réalisation de clusters Paramétrage de temps et des documents à prendre par moteur de recherche. Mais blocage quand demande beaucoup de documents.	Option: All (=AND), Any (=OR) "utilisation de + ou - si recherche avancée"
Inference FIND!	WebCrawler, Yahoo, Lycos, Altavista, Infoseek, Excite	Oui et réalisation de clusters	AND et OR
ProFusion	Altavista, Infoseek, Lycos, Excite, WebCrawler, OpenText,	Oui Avec la vérification de la disponibilité du document (LINK)	Selon le moteur (AND, OR, NOT, NEAR,...)
Mamma	Yahoo, excite, Infoseek, Lycos, Webcrawler, Altavista, Hotbot,	Oui	OR ou AND (à paramétrer)
Metafind	Planetsearch, Excite, Altavista, Infoseek, Webcrawler	Non Duplication	Selon le moteur sélectionné
Search.verio.net	A2z, Excite, HotBot, Infoseek, Lycos, Webcrawler, Yahoo	Oui Avec des scores affichés	AND (variation selon le moteur) Amélioration avec recherche avancée
Findit	Altavista,Yahoo, Hotbot,WWWebster, DejaNews	Non	Selon le moteur sélectionné

En regardant le tableau 4.1, on constate que certains métamoteurs retravaillent des résultats obtenus à partir des moteurs et enlèvent des duplications, les combinent et affichent des résultats compilés. D'autres méta-moteurs ne combinent pas des résultats et ils donnent des résultats tels qu'ils ont été obtenus des moteurs de recherche, la plupart de ces méta-moteurs jouent le rôle d'une interface où on peut trouver des moteurs de recherche et choisir quel moteur on veut utiliser.

Le travail qui suit consiste à prendre ces méta-moteurs qui combinent des résultats et enlèvent des duplications et on va faire des comparaisons de ces métamoteurs par rapport au nombre de documents retournés en fonction des mots clés.

4.3.3. Le nombre de documents retournés par les différents *méta-moteurs*

Conditions de travail

Nous nous donnons comme objectif d'obtenir un maximum de documents retournés, c'est pourquoi pour chaque méta-moteur, nous nous mettons dans des conditions qui permettent de les obtenir, en prenant un petit temps de lire des paramètres à changer à cette fin (logique OR, "*timout*" élevé).

Nous allons commencer avec un seul mot clé et puis on va augmenter le nombre de mots clés. Le tableau 4.2 montre les différents mots clés utilisés et la figure 4.1 illustre la variation du nombre de documents restitués par les différents *méta-moteurs* en fonction du nombre de mots clés.

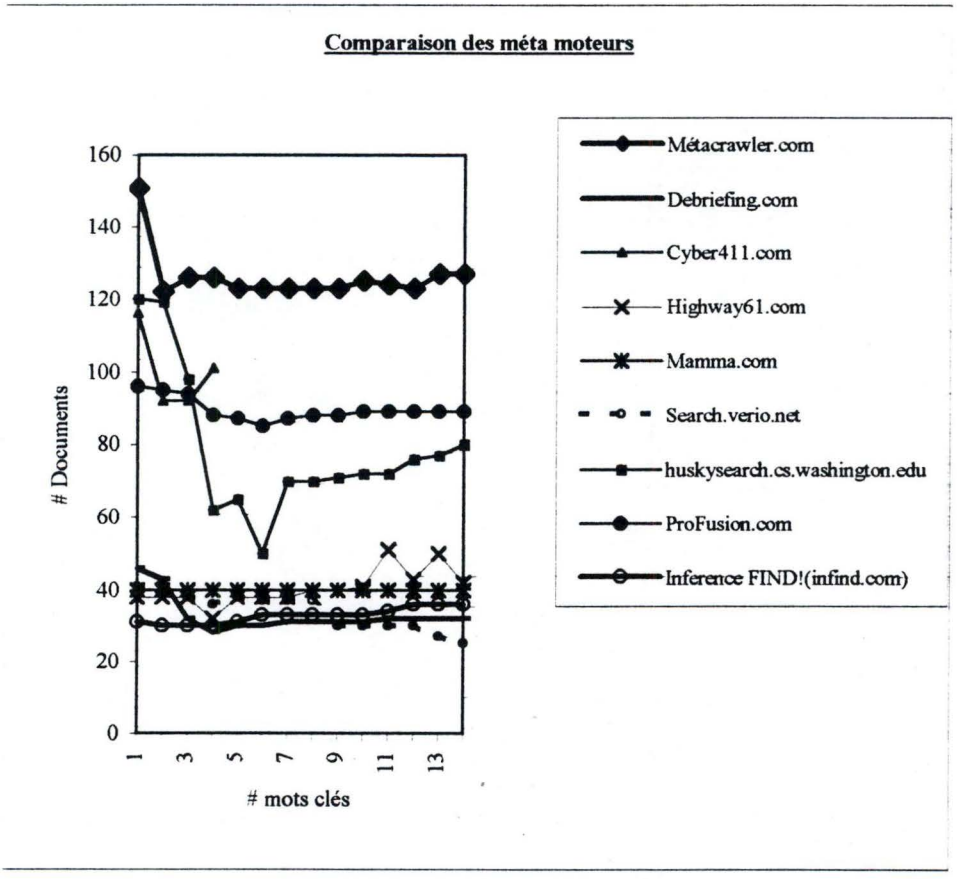


Figure 4.1: Le nombre de documents retournés par les méta-moteurs en fonction du nombre de mots clés

Sur la figure 4.1, on constate que :

- plus le nombre de mots clés augmente, plus le nombre de documents restitués ne semblent pas augmenter. Ceci peut être dû au fait que les mots clés utilisés sont sémantiquement équivalents, et par conséquent, on les trouve dans les mêmes documents;
- le méta-moteur *Cyber411.com* ne supporte pas plus de quatre mots clés;
- lesméta-moteurs *Metacrawler.com, ProFusion.com, huskysearch.cs.washington.edu* donnent un nombre de documents plus élevé par rapport aux autres méta-moteurs, mais le méta-moteur *hyskysearch.cs.washington.edu* a une très forte variation et puis une certaine augmentation.

Tableau 4.2 : Légende de la figure 4.1

Nombre de mots clés	**Mots clés utilisés**
1	Inulin
2	Inulin oligofructose
3	Inulin oligofructose raftilose
4	Inulin oligofructose raftilose fructoligosaccharide
5	Inulin oligofructose raftilose fructoligosaccharide raftiline
6	Inulin oligofructose raftilose fructoligosaccharide raftiline polyfructose
7	Inulin oligofructose raftilose fructoligosaccharide raftiline polyfructose fructan
8	Inulin oligofructose raftilose fructoligosaccharide raftiline polyfructose fructan polyfructan
9	Inulin oligofructose raftilose fructoligosaccharide raftiline polyfructose fructan polyfructan actilight
10	Inulin oligofructose raftilose fructoligosaccharide raftiline polyfructose fructan polyfructan actilight fructafit
11	Inulin oligofructose raftilose fructoligosaccharide raftiline polyfructose fructan polyfructan actilight fructafit inuline
12	Inulin oligofructose raftilose fructoligosaccharide raftiline polyfructose fructan polyfructan actilight fructafit inuline fructo-oligosaccharide
13	Inulin oligofructose raftilose fructoligosaccharide raftiline polyfructose fructan polyfructan actilight fructafit inuline fructo-oligosaccharide inulo-oligosaccharide
14	Inulin oligofructose raftilose fructoligosaccharide raftiline polyfructose fructan polyfructan actilight fructafit inuline fructo-oligosaccharide inulo-oligosaccharide inulooligosaccharide

Chapitre 5 :

5. Amélioration de la qualité de références fournies par les *méta moteurs* de recherche sur le Web

5.1. But de l'étude

L'étude sur l'amélioration de la qualité de références fournies par les méta-moteurs de recherche sur le Web va se faire par la procédure suivante :

- Un expert du domaine choisi est chargé, tout d'abord, d'élaborer une "table des matières" de ce domaine (évidemment restreint) pour lequel on veut fournir une base de références. L'expert doit accompagner cette classification d'un premier jeu de critères permettant à un "scooter" d'aller chercher les informations pertinentes au domaine et disponibles sur l'Internet.
- Un premier filtrage statistique intervient ici et consiste à l'analyse des documents retournés par le "scooter" pour en calculer la proximité avec les mots clés composant la requête introduite. Ce calcul se base sur des méthodes statistiques utilisées en *Information Retrieval* . Ce filtrage pourra ainsi intégrer d'autres critères tels que des contraintes sur la taille du document, la présence d'un mot dans le titre ou d'images, le type de fichier, la nouveauté du document, etc.
- L'expert est ensuite chargé de réévaluer ses critères en vue de maximiser le *rappel* de références pertinentes au domaine étudié. L'expert donne un avis positif ou négatif sur un document. Le système propose des mots clés à ajouter ou à retirer pour reformuler la requête. Ceci est bien sûr un processus itératif (*relevance feedback*) qui devrait mener à un résultat final consistant.
- Une fois que le processus décrit ci-dessus est achevé, nous avons une base ordonnée de références pertinentes à un sujet choisi. L'avantage d'une base de références sur une base de documents est que ceux-ci ne sont que des "pointeurs" vers de l'information qui peut être modifiée par son propriétaire légitime. Ce système est plus dynamique que le premier.

- Les résultats de ce processus sont dès lors convertis en fichiers au format HTML, qui ne demandent aucun nouveau logiciel pour être consultables. Un travail d'ergonomie sera ici nécessaire pour faciliter l'utilisation de cette base de références par le grand public. L'interface devra être particulièrement soignée.

Chapitre 6 :

6. Analyse et spécification

6.1. Introduction

Avant d'entamer l'analyse et la spécification de différents traitements à réaliser pour l'amélioration de la qualité de références fournies par les méta-moteurs de recherche sur le Web, il semble utile de donner d'abord quelques explications de certaines notions qui seront traitées dans ce travail. C'est pourquoi, nous allons décrire les notions de l'hypertexte, du HTML et son environnement, de formulaire sur le Web et CGI.

6.2. L'hypertexte

On pourrait définir (P. Chaleat *et al.*,1997) l'hypertexte comme un système qui relie des liens activables des éléments d'informations. Dans le concept du World Wide Web, on devrait préférer au terme hypertexte le terme de *hypermedia* puisque l'on va pouvoir accéder au texte mais aussi à l'image, au film, et au son. L'hypertexte introduit donc une notion importante de **liens**.

Un lien définit (P. Chaleat *et al.*,1997) une relation entre deux éléments d'information :

- un élément désignable à l'écran à l'aide de la souris : une lettre, un mot, un groupe de mots, une image ou une portion d'image. Cet élément est généralement signalé à l'attention du lecteur par un attribut visuel;
- sa cible, qui peut être du texte "plat", un autre document hypertexte, une image, un film ou un son.

On distingue plusieurs types de liens :

- **lien interne** : c'est un lien qui référence à une partie se trouvant à l'intérieur du même document;

- **lien externe** : référence vers un autre document;
- **lien exécutable**: c'est un lien externe aussi mais ce lien déclenche un programme informatique de traitement de données en réponse à une action effectuée par le lecteur de manière interactive.

6.3. HTML et son environnement

6.3.1. Qu'est ce que HTML?

Selon P. Chaleat *et al.*, (1997) et J. Baldi (1997), le HTML (HyperText Markup Langage) est un langage simple utilisé pour créer des documents hypertextes pour le World Wide Web. Ce langage de définition hypertexte a été standardisé par l'IETF(Internet Engeneering Task Force). Ce langage qui est utilisé depuis 1990 dans le cadre W3 du CERN est une implémentation relativement simple de SGML (Standard Generalized Markup Langage).

6.3.2. Dans quel contexte s'exécute HTML ?

Le cadre d'exécution de Web et de son langage HTML est celui d'un modèle client-serveur véhiculé sur le réseau informatique comme le réseau Internet.

- **Le client :** c'est un logiciel de consultation s'exécutant sur tous les types de plate-forme (Macintoch, station Unix, PC). Il dialogue avec un serveur selon un protocole spécifique HTTP (HyperText Transfert Protocol) qui va lui fournir l'information structurée en code HTML. C'est ce client qui a en charge l'exécution de ce code (on parle aussi du langage source) et qui produira l'affichage du document. Le client peut s'entourer d'éléments périphériques pour afficher des documents dont il n'est pas capable d'utiliser le format. C'est le cas pour les films, le son et divers formats d'images dont Postscript fait parti. Les clients les plus connus sont Mosaic, Netscape, Internet Explorer, ...

- **Le serveur :** ce logiciel est en permanence à l'écoute des requêtes que vont formuler les clients. Il en vérifie la validité, s'assure que la demande émane d'un client autorisé à accéder au document, et lui envoie finalement les données

demandées avant de clore la connexion. A signaler que dans le World Wide Web lorsque on se connecte à un serveur, la connexion avec celui-ci ne dure que le temps de l'envoi de la page demandée, pour chaque page demandée au même serveur, on recommence la procédure de connexion. Un serveur peut être sollicité à chaque instant. Il est donc important que la machine qui l'accueille soit toujours disponible et présente sur le réseau. Bien que l'on trouve des logiciels serveurs sur toutes les plates-formes (y compris Macintosh et PC), dans la majorité des cas ceux-ci seront implantés sur des stations de travail Unix ou VMS principalement. La puissance de ces types de machine permet d'accepter simultanément un grand nombre de requêtes. Les logiciels serveurs couramment utilisés sont ceux fournis gratuitement par le CERN ou par NCSA. Dans la terminologie Unix, on les nomme *démons HTTPD* (HyperText Transfert Protocol Daemon) (P. Chaleat *et al.*, 1997).

6.3.3. Le protocole d'adressage des documents - URL

Un document sur le réseau Internet est identifié par un seul moyen unique URL (Uniform Ressource Locator).

Un URL va être composé par des éléments suivants :

- le protocole d'échange entre le client et le serveur, HTTP mais d'autres protocoles existent, on peut citer FTP, Telnet,... ;
- l'adresse Internet du serveur qui diffusent les documents. Cette adresse est unique sur tout le réseau, c'est l'adresse TCP/IP de la machine. Elle a la forme d'une suite de nombres telle que 138.48.0.0. Comme ces nombres ne sont facilement mémorisables, un annuaire DNS (*Domain Name System*) résout généralement la relation adresse numérique; nom symbolique de la machine/nom du domaine (exemple à l'Institut d'Informatique on a http://www.info.fundp.ac.be);
- l'arborescence des répertoires (le chemin) qui conduit au document;
- le nom du document qui aura toujours l'extension .**html (ou htm)**

La syntaxe que l'on rencontre couramment pour un URL d'un document est :

Protocole://nom_du_serveur/repertoire/sous-repertoire/nom_du_document

Exemple d'URL: http://www.info.fundp.ac.be/~gnzamura/Index.html

6.3.4. Structure du document HTML

Sans entrer dans les détails de la programmation du langage HTML, la figure 6.1 présente l'ossature type que l'on trouve toujours dans un document HTML

```
<HTML>
<!--Squelette HTML -->
<HEAD>
        <TITLE>Emplacement du title</TITLE>
</HEAD>
<BODY>
        Document
        ....
</BODY>
</HTML>
```

Figure 6.1: Squelette d'un document HTML

La figure 6.1 illustre la structure d'un document HTML avec un marquage minimal. Sur cette figure, on constate que tout document HTML commence par la balise (ou la marque) <HTML> et se terminer par la marque </HTML>. Le titre du document sera généralement inclus dans une zone de prologue ou d'en tête identifiée par la balise <HEAD> et </HEAD>. Les marques <TITLE> et </TITLE> sont utilisées pour identifier le titre du document HTML au sein du navigateur (ou browser). La différence entre les balises <HEAD> et <TITLE> est que la balise <HEAD> indique le titre sur la page et que la marque <TITLE> indique le titre dans la barre de titres du navigateur, et c'est ce titre qui est repris pour la gestion des signets ou bookmarks et quand on est dans les systèmes où on peut iconifier la fenêtre, c'est ce titre qui va être pris comme nom de l'icône Tout le reste du document HTML doit se trouver entre les balises <BODY> </BODY>.

Les commentaires dans code source du document HTML sont placés entre les balises <!-- et-- >, ces commentaires ne seront pas affichés par le navigateur (P. Chaleat *et al*, 1997; Baldi,1997).

A part ces balises obligatoires illustrées sur la figure 6.1 du document HTML, d'autres balises structurelles existent surtout pour la mise en forme d'un document HTML, c'est pourquoi par exemple, on peut rencontrer des balises pour créer des tables, des listes, de styles,...

Dans ce travail, nous n'avons pas voulu entrer dans les détails de ces balises.

Le document HTML souvent contient les liens hypertextes, et ces liens sont signalés par la balise <HREF= suivi de l'adresse du document référencé.

6.4. Formulaire et CGI

Un formulaire (*form*) est un document HTML, il est donc une réponse apportée par le langage HTML pour générer sur l'écran des zones de dialogue avec le client. Comme dans un formulaire papier on va pouvoir réaliser des zones dans lesquelles on entrera un texte, des cases à cocher, des listes de choix,...

Des formulaires ne seront jamais plus que des fiches que l'on remplit; il faudra toujours un élément approprié pour traiter l'information qui y est saisie. Cet élément sera presque toujours un programme (scripts) s'exécutant dans un espace du serveur réservé à cet effet: le CGI (ou *Common Gateway Interface*). On a donc besoin de cette passerelle et d'interface parce que le plus souvent le programmeur devra écrire un script permettant d'établir le dialogue entre le navigateur soumettant son formulaire et un autre programme accessible depuis la machine serveur comme une base de données par exemple.

6.4.1. Le principe du formulaire

On va décrire à l 'aide des balises HTML les zones dans lesquelles l'utilisateur va remplir son formulaire. Chacune de ces zones sera identifiée par un nom symbolique, et lorsque le formulaire est envoyé au programme d'exploitation, celui-ci reçoit l'identificateur de chaque zone et la valeur saisie.

Par exemple (P. Chaleat *et al.*, 1997), on décrit un champ dans lequel l'utilisateur du formulaire va taper son nom et son prénom; on appelle cette zone du nom symbolique *identité*. L'utilisateur frappe dans ce champ *David Martin*. Le script (le programme CGI) reçoit la donnée *identite = David Martin.*

6.4.2. Description des balises de formulaire

Un formulaire étant un document HTML, il doit contenir les balises de bases vues précédemment constituant un document HTML. Un formulaire sera identifié par l'ajout entre les balises <BODY> et </BODY> du document HTML d'autres balises spécifiques aux formulaires.

La balise <FORM>

La balise <FORM> débute un formulaire. Entre <FORM> et </FORM> se situent toutes les balises générant les divers boutons, les cases à cocher, entrées de textes, etc, mais aussi du texte HTML standard permettant "d'habiller", ou "auto-documenter" le formulaire. Deux attributs complètent obligatoirement cette balise :

- Attribut METHOD : cet attribut peut prendre deux valeurs "GET" ou "POST" indiquant la façon de transfert des données acquises vers le script. Quand on utilise la méthode GET, les données envoyées au script sont incluses dans l'URL de l'application sous forme d'une chaîne de requête tandis qu'avec la méthode POST les données sont envoyées à l'application sous forme de contenu via l'entrée standard d'une application CGI.
- Attribut ACTION : la fonction de l'attribut ACTION est d'indiquer l'action à prendre (le script à exécuter) lorsque l'on clique sur le bouton de soumission. L'attribut ACTION définit donc l'URL du script permettant d'exploiter le formulaire.

La syntaxe générale d'ouverture d'un formulaire est donc :

<FORM METHOD=type_de_méthode ACTION=URL_du_script>

Exemple <FORM METHOD = "POST" ACTION = /cgi-bin/Inscription>.

La balise <INPUT>

Cette balise sert à définir des champs pour entrer un texte et des boutons permettant de choisir des options.

L'attribut TYPE associé à la balise <INPUT> permet le choix de l'élément d'entrée :

- le type SUBMIT déclenche l'envoi du formulaire vers le script;
- le type RESET permet d'effacer les données déjà entrées;
- le type PASSWORD permet de saisir un mot de passe de façon confidentielle;
- le type CHECKBOX permet de faire un bloc de boutons permettant un choix multiple d'options (fonction logique *et)*;
- le type RADIO permet de faire un bloc de bouton permettant un choix exclusif parmi plusieurs options (fonctions logique *ou exclusif)*.

Les balises <SELECT> </SELECT> et <OPTION>

Les balises **<SELECT> </SELECT>** permettent de générer des listes à choix simple (*ou exclusif*) ou à choix multiple (*et*). Elle se programme comme une liste où chacun des items est spécifié par la balise <OPTION>. De la présence de l'attribut SIZE va dépendre la présentation de la liste. Si la valeur donnée à l'attribut SIZE est inférieure à 2 ou si l'attribut SIZE est omis, la liste s'affiche comme un menu déroulant (*pop-list*). Dans le cas contraire, la liste s'affiche dans une fenêtre à ascenseurs. La valeur donnée à l'attribut SIZE donne alors le nombre de lignes visibles dans la fenêtre. L'option *choix multiple* découle de la présence de l'attribut MULTIPLE.

La balise <TEXTAREA>

La balise <TEXTAREA> permet de créer une fenêtre avec des ascenseurs horizontaux et verticaux dans laquelle on pourra saisir du texte. Les attributs ROWS (lignes) et COLS (colonnes) permettent de spécifier la taille de cette fenêtre.

Par ailleurs, il convient que le script connaisse de quel élément du formulaire lui provient une donnée. L'attribut NAME permet d'effectuer un identificateur à chacun des éléments du formulaire.

L'attribut VALUE, quant à lui, permet :

- de spécifier un texte à écrire sur les boutons *reset* et *submit* (informations visuelle);
- de préremplir un champ *texterea (*information visuelle);
- d'attribuer une valeur à l'option pour les boutons *radio* et *checkbox (*information pour le script). La figure 6.2 illustre un exemple d'un formulaire simple.

```
<HTML>
<HEAD><TITLE>Formulaires</TITLE></HEAD>
<BODY>
<b>
<FORM METHOD="post" ACTION="/cgi-bin/form1">
<H4>Choisissez les chapitres &agrave; imprimer</H3>
<SELECT NAME="Test" SIZE=6 MULTIPLE>
<OPTION> Introduction
<OPTION> Technique de l'hypertexte
<OPTION> L'adressage des documents
<OPTION> La structure du langage
<OPTION> Les balises
<OPTION> L'accentuation
<OPTION> Les tableaux
<OPTION> Les images
<OPTION> Les images cliquables
<OPTION> Les formulaires
<OPTION> La s&eacute;curit&eacute;
</SELECT><p>
<INPUT TYPE="submit " VALUE="Imprimer">
<INPUT TYPE="reset" VALUE="Tout d&eacute;s&eacute;lectionner">
</FORM>
</b>
</BODY>
</HTML>
```

Figure 6.2 : Exemple du code source d'un formulaire simple (P. Chaleat *et al.*, 1997)

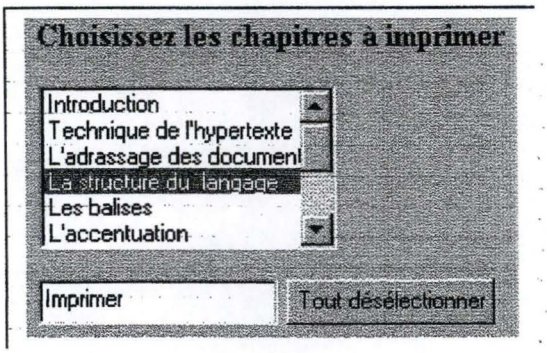


Figure 6.3 : Exemple de formulaire dans le navigateur *Internet explorer.*

La figure 6.3 illustre l'aperçu du formulaire dans le navigateur Internet explorer. Par ailleurs sur la figure 6.2 on y voit quelques marquages tels que et et <p> qui sont des marquages de styles et de mise en pages.

6.4.3. Document HTML interactif

L'évolution du langage HTML avec l'arrivée des nouveaux langages de programmation comme Java et Javascript ont permis de voir l'apparition des documents HTML interactifs. Par exemple; pour le remplissage du formulaire, le serveur peut envoyer chez le client un formulaire contenant un programme lui permettant de corriger ses erreurs avant l'envoi de ce formulaire. Ces programmes qui s'exécutent chez le client en langage JAVA on les appelle les APPLETS. Le problème qui subsiste dans ces types de programme concerne le sécurité au niveau du client (ANUP K. GHOSH, 1998).

6.5. La programmation CGI

6.5.1. Introduction

Comme l'on a déjà vu jusqu'à présent, le code HTML est en général inséré dans des fichiers. Le dialogue entre le client (Nescape, Internet explorer, ...) et le serveur (Netscape, Apache...) se déroule alors de la façon suivante :

- le client demande un fichier au serveur ;

- le serveur envoie l'information contenue dans ce fichier (du code HTML) au client;
- le client se charge de la mise en page.

Les fichiers contenant du code HTML pouvant être accédés à tout moment par les clients, il ne faut donc jamais les modifier. En effet, un client qui fait une requête sur un fichier en cours de modification risquerait d'avoir une version non cohérente de ce fichier. On constate donc que ce type de fonctionnent ne permet pas de créer des documents dynamiques. Par exemple la création d'un document contenant la date ou la création d'un document résultat d'une requête sur une base de données.

L'idée de la programmation des CGI (*Common Gateway Interface*) est de construire un document HTML correspondant à un lien hypertexte au moment même où l'on clique sur ce lien. Le document est envoyé au client au fur et à mesure de sa construction sans jamais être stocké dans un fichier (P. Chaleat *et al*, 1997).

Cette construction dynamique de document est réalisée par des liens exécutables. Le client indique le nom d'un fichier, toujours à l'aide d'une URL, non pas pour en recevoir le contenu mais pour demander l'exécution de ce fichier sur le serveur. Ce dernier exécute le programme indiqué et renvoie au client la sortie standard de ce programme (c'est à dire ce que l'on aurait obtenu à l'écran en lançant le programme à la main sur une ligne de commende). C'est cette sortie standard qui constitue le document HTML. On appelle des scripts *cgi,* tous les programmes lancés à partir de liens exécutables.

6.5.2. Emplacement des scripts cgi

Lors de la configuration du serveur Web, il est possible d'indiquer quels sont les répertoires susceptibles d'accueillir des scripts *cgi*. Lorsque le serveur reçoit une requête pour une URL référençant un fichier dans un tel répertoire, il considère que ce fichier est un script à exécuter; il tante donc de l'exécuter et de renvoyer sa sortie standard vers le client.

Dans la pratique, on utilise très souvent le répertoire par défaut créé lors de l'installation du serveur. Le nom et l'emplacement de ce répertoire peuvent varier d'un serveur à un autre. Dans le cas du serveur Apache de l'Institut d'Informatique de Namur, il s'agit du répertoire */cgi-bin.*

6.5.3. Les langages de programmation et la sécurité

Selon P. Chaleat *et al.* (1997) tout langage qui peut avoir une sortie standard peut être utilisé pour écrire des scripts *cgi*. Dans un environnement Unix on pourra utiliser des langages comme : C, Fortran, Perl, Tcl, ou encore tous les shells Unix. Sur les PC-Windows, on pourra utiliser *Visual C++*, *Visual Basic* et sur un Macintosh *Apple Script*

Dans la suite de ce travail, nous traiterons à part des apports des nouveaux langages comme Java pour les scripts. Ces apports sont très récents et jusqu'à présent ne sont pas encore généralisés dans le domaine de l'Internet, mais dans l'avenir, sûrement qu'ils seront généralisés vu leur facilité de programmation et le succès accru du langage Java qui est un langage surtout orienté vers l'Internet.

Classiquement, le langage le plus utilisé dans le monde Unix est le langage *Perl*, qui est u langage interprété. La langage C, qui est langage compilé, est également plus utilisé dans la programmation des scripts CGI. La compilation offre un plus au niveau de sécurité de la sécurité. La compilation permet de s'assurer que les sources d'un script *cgi* ne seront jamais accessibles par le réseau, même par mégarde, ce qui n'est jamais le cas avec les langages interprétés (P. Chaleat, 1997, ANUK K. GHOSH, 1998).

6.5.4. Quelques mots sur la sortie standard d'un scripts CGI

Le format

Comme on a pu le constater au point 6.5.1, un script *cgi* est tout simplement un programme qui écrit sur sa sortie standard. Pour que cette sortie standard puisse être interprété par le serveur et le client, elle doit cependant avoir un format spécial. La figure 6.4 présente un exemple de la sortie standard d'un script *cgi*. Cette sortie se compose de deux parties séparée par une ligne vide (un Carriage Return/Line Feed).

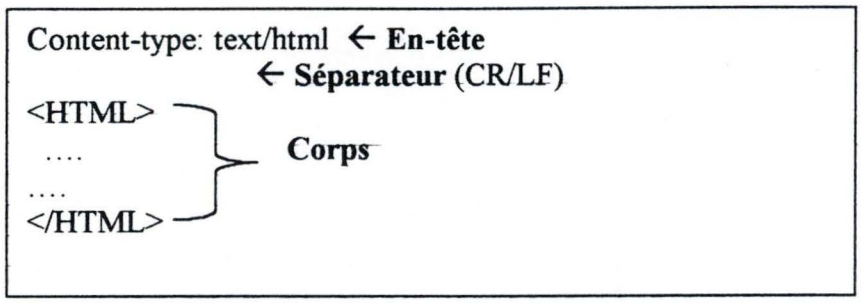


Figure 6.4 : Les parties de la sortie standard d'un script *cgi* (P.Chaleat, 1997)

La première partie de la sortie standard (figure 6.4.) d'un script *cgi* est **l'en-tête.** L'en-tête concerne des informations qui seront utilisées par le serveur pour construire l'en-tête HTTP de sa réponse au client. La seconde partie de la sortie standard du script *cgi* est le **corps.** Le corps est composé par des données constituant le document qui sera envoyé au client.

L'en-tête

Trois lignes peuvent apparaître dans l'en-tête. La ligne Content-type, la ligne Location et la ligne Status (P. Chaleat *et al.*, 1997).

Content-type *:* comme le script cgi doit générer des documents HTML dynamiques, un des principaux rôles de l'en-tête est d'indiquer le type des données que le script va crée. Un sript *cgi* peut générer tout types de documents reconnus par le client, comme par exemple le son, des images, ...Il est donc indispensable que le script *cgi* commence par indiquer le type des données qu'il va générer de façon que le client puisse savoir comment il faut le gérer. C'est le rôle de la ligne Content-type.

La syntaxe MIME (*Multi-purpose Internet Mail Extensions*) qui est une extension du protocole SMTP (*Simple Mail Transfer Protocol*) est utilisé pour indiquer le type des données: *Content-type: type/subtype.*

Le tableau 6.1 illustre les types les plus utilisées dans la programmation des scripts *cgi.*

Tableau 6.1 : Les types les plus courants dans la programmation des scripts *cgi* (P. Chaleat *et al.*, 1997)

type/subtype	**Description**
text/html	Le type le plus utilisé, il indique au client que les données doivent être interprétées comme des commandes HTML.
text/plain	Il indique au client que les données sont du texte plat, qui ne doivent pas être interprétées.
image/gif image/jpeg image/x-xbitmap	Ce sont les trois types d'images supportées en général par les clients WWW sans appel un *viewer* externe
audio/basic	C'est un type englobant tous les formats son .au et .snd.
Application/postscript	Il indique au client que les données sont au format postscript. En général, le client fait appel à un interpréteur externe pour afficher le document postscript

Location : la ligne **Location** dans l'en-tête permet à un script *cgi* de rediriger un client vers une URL donnée. Lorsque l'on utilise la ligne Location la sortie standard ne possède ni la ligne Content-type ni le corps puisque on référence un autre document qui existe déjà. La syntaxe est la suivante : *Location: URL.*

La ligne Location est particulièrement utile quand à l'issue du script *cgi* on désire se placer sur une URL existante (qui peut être le résultat d'une recherche...) et non générer un nouveau document (P. Chaleat *et al.*, 1997).

Status : cette ligne, rarement utilisée dans l'en-tête de la sortie standard du script *cgi*, indique le code de retour que le serveur doit envoyer au client lors du dialogue HTTP. La syntaxe est : *Status : code message* ;

Lorsque cette ligne n'est présente, le code de retour est *'200 OK'* ce qui se traduit dans le dialogue HTTP du serveur vers le client par la ligne *'HTTP 200 OK'*. Cependant les clients ne tiennent pas compte systématiquement de la valeur de ce code (P. Chaleat *et al.*, 1997).

Le corps

Après l'en-tête qui constitue la première partie de la sortie standard du script *cgi*, (figure 6.4), la seconde partie de la sortie standard du script *cgi* est le **corps.** Le corps contient les données du document qui doivent correspondre avec le type MIME annoncé dans le Content-type de l'en-tête.

Non Parsed Header

On a bien vu que lorsque un script *cgi* est exécuté, la sortie standard est redirigé vers le serveur Web qui, à partir de l'en-tête qu'il reçoit, crée l'en-tête HTTP qui va être envoyé au client. Cependant il est possible de générer directement l'en-tête HTTP complète à partir du script *cgi*. Le serveur n'a alors plus besoin de *parser* la sortie standard du script, ce qui évite un surplus de travail. Par convention, pour indiquer au serveur que le script se charge de la génération de l'en-tête HTTP, le nom du script doit être préfixé par *nph* (pour *Non Parsed Header*). Par exemple : */cgi-bin/nph-nom_script* (P. Chaleat *et al.*, 1997).

6.5.5. Les variables d'environnement et les CGI

Lorsque un *démon httpd* lance un script, il positionne une série de variables d'environnement. Ces variables jouent un rôle capital dans la programmation *cgi*. Les variables d'environnements vont permettre au développeur d'accéder à de nombreuses informations, concernant le serveur, le client, la machine qui exécute le client etc. DINANT J.M(1999) parle de "*bavardage sur l'Internet!*" Il s'agit d'un des moyens de communication entre le serveur et le script *cgi*.

Voici une liste de quelques variables d'environnement créées par le démon avant d'exécuter un script *cgi* (P. Chaleat *et al.*, 1997).

Variables ne dépendant pas de la requête

- SERVER_SOFTWARE : nom et version du démon httpd qui tourne le script *cgi;*
- SERVER_NAME : nom de la machine qui tourne le démon httpd ou IP si cette machine n'a pas de nom IP;
- GATEWAY_INTERFACE : niveau de la passerelle *cgi* du serveur Web (Format: CGI/version).

Variables spécifiques à une requête donnée

- SERVER_PROTOCOL : nom et version du protocole utilisé par la requête en cours de traitement (format: protocole/version);
- SERVER_PORT : numéro du port (au sens IP) vers lequel la requête a été envoyée;
- REQUEST_METHOD : méthode avec laquelle la requête a été effectuée. Cette variable permet de savoir si un formulaire est gérée par la méthode POST ou GET;
- SCRIPT_NAME : nom du script à partir de la racine du serveur Web;
- REMOTE_HOST : nom de la machine à l'origine de la requête. Si cette machine n'a pas de nom, le serveur renseigne uniquement la variable REMOTE_ADDR;
- REMOTE_ADDR : adresse IP de la machine à l'origine de la requête;
- AUTH_TYPE : variable permettant l'authentification du client par le serveur quand le script est protégé, cette variable indique la méthode utilisée pour valider l'utilisateur;
- REMOTE_USER : variable indiquant le nom de l'utilisateur lors de l'authentification de l'utilisateur par le serveur quand le script appelé est protégé;
- REMOTE_IDENT : variable non supporté par tous les serveur, cette variable permet quand elle est positionnée de savoir le nom de l'utilisateur logué sur la machine à l'origine de la requête;
- CONTENT_TYPE : pour les requêtes qui véhiculent les données cette variables indique le type des données envoyées par le client. Le format utilisé est le format MIME;

- CONTENT_LENGTH : variable permettant de connaître le nombre de caractères à lire sur l'entrée standard lors du traitement d'un formulaire par la méthode POST.

Variables de passages de paramètres au script

En regardant la figure 6.5, on peut constater des variables PATH_INFO et QUERY_STRING.

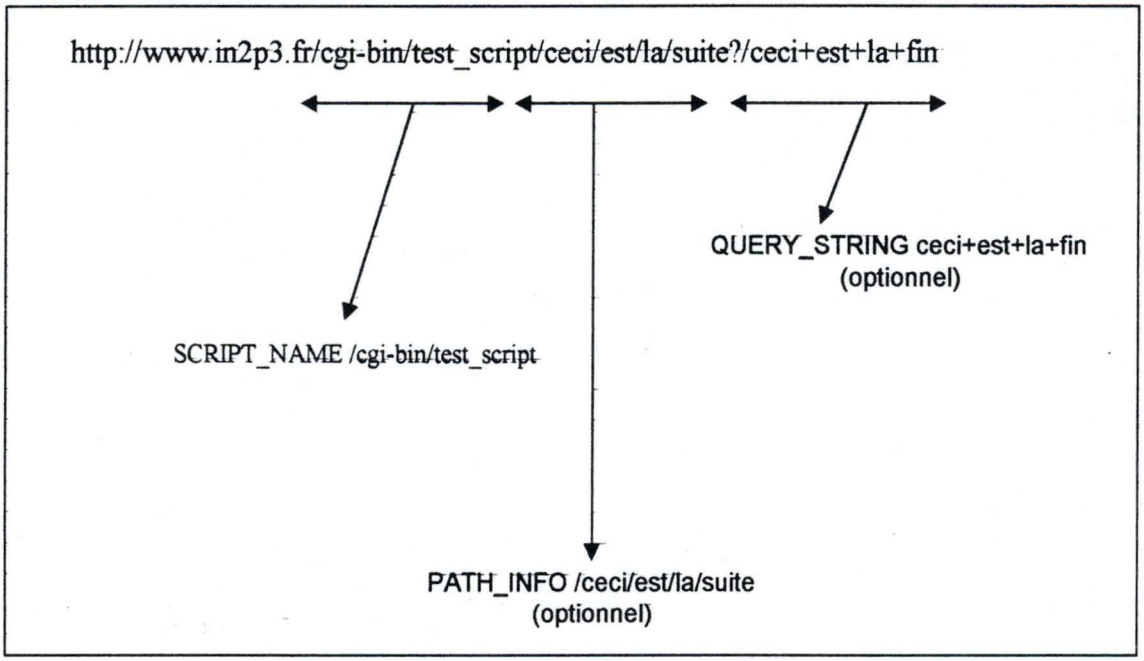


Figure 6.5 : URL référençant un script *cgi* (P. Chaleat *et al.*, 1997)

- PATH_INFO : cette variable se définit comme une chaîne de caractères qui commencent après le nom du script et qui se termine avant le caractère '?' ou à la fin de l'URL s'il n'y a pas de '?';

- QUERY_STRING : c'est la chaîne de caractères qui suit le premier '?' après le nom du script et se termine à la fin de l'URL.

Dans le cas de la figure 6.5, PATH_INFO=/ceci/est/la/fin et QUERY_STRING = ceci+est+la+fin (P. Chaleat *et al.*, 1997).

La chaîne de caractères de QUERY_STRING a la particularité d'être *URL-encodé*. C'est à dire que les espaces y sont remplacés par des + et les caractères spéciaux par leur valeur hexadécimale sous forme %xx. Cet encodage est fait automatiquement lorsque la chaîne est générée par le client. Par contre, si la chaîne est ajoutée à la main, elle doit être encodée auparavant.

La variable QUERY_STRING est utilisé pour envoyer les données au serveur Web par le client quand la méthode du formulaire est GET.

Autres variables

Sans entrer dans les détails certaines variables existent et surtout peuvent être utilisées pour divers effets, y compris pour des effets purement commerciaux. En effet, sachant les outils que disposent le client, on peut lui envoyer des publicités qui ont en rapport avec le profil de sa machine et des logiciels qui tournent sur celle-ci, et ça peut même aller plus loin concernant la confidentialité et la sécurité des machines se trouvant sur le réseau Internet (J.M. DINANT, 1999).

- PATH_TRANSLATED : il s'agit de la variable PATH_INFO préfixé par la racine du serveur Web;
- HTTP_ACCEPT : cette variable liste des types MIME acceptés par le client qui a fait la requête;
- HTTP_USER_AGENT : information concernant le client qui a fait la requête (format: logiciel/version librairie/version)
- HTTP_USER_AGENT : il s'agit de la marque du navigateur, de son nom et le numéro de sa version. Souvent la langue utilisée par le programme de navigation et la marque du processeur principal de la machine utilisée sont transmises par cette variable;
- HTTP_REFERRER : le référant est une variable créée par le programme de navigation lors de la mise en œuvre des hyperliens;

- HTTP_LANGUAGE : dans certains navigateurs, il est possible d'encoder les langues acceptées par l'internaute. Ces renseignements seront systématiquement transmis via le programme de navigation lors de chaque requête HTTP (J.M. Dinant, 1999);
- HTTP_COOKIES : lorsqu'un navigateur reçoit la variable HTTP_COOKIES d'un site Internet, il stocke par défaut systématiquement son contenu sur le disque dur de l'internaute. Par la suite, cette valeur du cookie sera automatiquement transmise dans l'entête HTTP lors de chaque requête (explicite ou non) faite au site qui a stocké le cookie dans un premier temps. Le site peut alors modifier à nouveau cette valeur.. Mieux qu'une adresse IP dont la valeur peut être attribuée dynamiquement par les fournisseurs d'accès, les cookies permettent de marquer d'une manière invisible mais très efficace les machines utilisées par les internautes (J.M. Dinant, 1999).

6.6. Apports de la technologie Java

6.6.1. Introduction

Selon E. Puybaret (1998), Java est un langage de programmation développé par Sun Microsystems. Il n'a que quelques années de vie (les premières versions datent de 1995 !), et pourtant il a réussi à intéresser et intriguer beaucoup de développeurs à travers le monde. Et pourquoi donc?

La réponse, selon le même auteur, est vaste et forcément sujet à polémiques. En voici les principaux avantages :

- c'est un langage orienté objet dérivé du C, mais plus simple que le C++;
- il est multi-plateforme : tous les programmes devraient tourner sans modification sur toutes les plate-formes où existe Java;
- il est doté en standard d'une riche bibliothèque de classes, comprenant la gestion des interfaces graphiques (fenêtres, boites de dialogue, contrôles, menus, graphisme), la programmation multi-threads (multitâches), la gestion des exceptions, les accès aux fichiers et au réseau (notamment Internet),...

Les deux derniers points ont contribué à utiliser ce langage pour développer des applets, qui sont des applications qui peuvent être téléchargées via Internet et exécutées dans un navigateur sur n'importe quelle plate-forme. Ainsi, une page statique HTML peut s'enrichir de programmes qui lui donneront un comportement dynamique comme vu précédemment au point 6.4.3.

A part des applets qui s'exécutent chez le client, le langage Java permet la programmation des servlets qui sont des applications qui tournent sur le serveur. Les servlets sont d'un intérêt croissant vu leur capacité de résoudre tous les problèmes liés à la programmation de CGI.

6.6.2. Servlets

Selon F. Delonnoy (1998), les servlets sont aux serveurs ce que sont les applets aux browsers. Les servlets correspondent à des programmes Java normaux qui utilisent des modules supplémentaires (ainsi que les classes et les méthodes associées) figurant dans l'API des servlets Java. Les servlets s'exécutent sur une machine de serveur Web à l'intérieur d'un serveur compatible avec Java. Ils permettent l'extension des fonctions du serveur.

Une servlet peut être chargée automatiquement lors du démarrage du serveur Web. Elle peut également être chargée lorsque le premier client demande les services de la servlet. Une fois chargée, les servlets restent actives dans l'attente d'autres requêtes du client (F. Delonnoy, 1998),.

Les servlets permettent l'extension des fonctions du serveur grâce à la création d'un environnement de prestation de services de requête/réponse via le Web. Un client envoie une requête au serveur. Ce dernier transmet à la servlet les informations relatives à la requête. La servlet crée ensuite une réponse que le serveur renvoie au client.

Dans la mesure où il s'agit d'un programme Java, la servlet peut utiliser toutes les fonctions du langage Java lors de la création de la réponse. La servlet peut également communiquer avec des ressources externes telles que des fichiers ou des bases de données, ou avec d'autres applications (également écrites en langage Java ou dans d'autres langages), afin de créer la réponse et éventuellement de sauvegarder des informations relatives à l'interaction requête/réponse.

La réponse envoyée au client peut donc être une réponse dynamique et unique conçue pour une interaction particulière et non une page HTML statique existante.

6.6.3. A quoi servent les servlets ?

Les servlets exécutent un grand nombre de fonctions, par exemple (F. Delonnoy, 1998) :

- une servlet peut créer et renvoyer une page Web HTML complète dont le contenu dynamique dépend de la nature de la requête du client;
- une servlet peut simplement créer une partie d'une page Web HTML qui est intégrée à une page HTML statique existante;
- une servlet peut communiquer avec d'autres ressources du serveur, y compris des bases de données, d'autres applications Java et des applications écrites dans d'autres langages;
- une servlet peut traiter les connexions avec plusieurs clients en acceptant les données en entrée de plusieurs clients et en diffusant à ces derniers des résultats

6.6.4. Les avantages des servlets ?

Les servlets (F. Delonnoy, 1998) sont les produits d'interconnexion entre Web et base de données qui:

- sont indépendants des OS (Unix ou NT) ;
- sont indépendants des serveurs Web (Apache, Netscape, ...) ;
- peuvent produire de l'HTML côté client (notamment pour la consultation de la base), sur la base de HTTP ;
- font mieux que le CGI en prenant en charge les connexions des utilisateurs en multi-thread automatiquement.

6.6.5. Les différences entre les scripts CGI et les servlets

Selon (F. Delonnoy, 1998), la plus grosse différence entre les CGI et les servlets est la performance. Il n'y a qu'une seule machine virtuelle Java qui tourne sur le serveur, et la servlet est placée en mémoire une fois qu'elle est appelée. Les servlets résident en mémoire, de ce fait leur exécution est très rapide. L'information statique peut être partagée par plusieurs invocations de la servlet, vous autorisant donc à partager cette information entre plusieurs utilisateurs. Les servlets sont modulaires, chaque servlet peut accomplir une tache spécifique et ainsi on peut les rendre communicantes.

6.6.6. Mise en œuvre des servlets

Pour mettre en œuvre des servlets, il faut posséder Servlet Development Kit 1.0. Ce produit est inclus dans le Java Development Kit 1.2. Il contient un interpréteur de servlet pour tester les servlet, les sources du package Servlet, et des exemples pour les serveurs Web Netscape, Microsoft, et Apache (F. Delannoy, 1988).

6.6.7. Passage des informations à une servlet par un formulaire HTML au moyen d'une méthode POST ou GET

Les informations contenues dans un formulaire HTML sont transmises au serveur grâce aux variables d'environnement (point 6.5.5). Le langage Java facilite la récupération des ces variables d'environnement par les classes et les méthodes que l'on trouve dans l'API Java Servlets (voir Annexe I).

La transmission des données contenues dans un formulaire HTML à un serveur via les servlets est un domaine qui a été bien mis au point dans cette API Servlets Java. Par cette API, on n'a plus besoin d'écrire des scripts CGI en langage de bas niveau comme Perl, C, (L. Goffinet, 1999), tout a été prévu à cet effet.

6.7. Analyse des documents retournés par les méta-moteurs

Maintenant que l'on a défini quelques notions concernant les documents HTML, nous allons faire une analyse des documents que l'on obtient quand on interroge des méta-moteurs.

Pour interroger un méta-moteur, le client se connecte d'abord à l'adresse URL de ce méta-moteur. A partir du formulaire reçu du méta-moteur, le client compose la requête et avec le bouton "*submit*"(envoyer) le client soumet le formulaire rempli au serveur du méta moteur. Ce dernier se charge de trouver des documents répondant à la requête et ces documents sont retournés au client. Les figures 6.7 et 6.8 illustrent la présentation d'une partie des documents retournés par les méta-moteurs *verio.com* et *Highway61.com*. pour le mot clé "inuline".

Comme on peut le constater sur les figures 6.7 et 6.8, les résultats obtenus d'une requête se présentent sous forme d'un document HTML qui contient les liens hypertextes signalant les titres des documents répondant à la requête et leurs adresses URLs . Quand le lecteur veut lire ou télecharger un document parmi l'ensemble des documents retournés, il doit cliquer sur le lien du document qui l'intéresse.

Avec une analyse approfondie des figures 6.7 et 6.8, on peut constater que certains méta-moteurs donnent la pertinence du document sans dire comment ils calculent ce score. Par ailleurs, certains méta-moteurs donnent un résumé du document et d'autres non. Quelques méta-moteurs affichent le titre du document, son résumé sans afficher son URL.

A signaler que presque pour tous les méta-moteurs, le document HTML contenant les références des documents répondant à la requête contient toujours des liens de publicité.

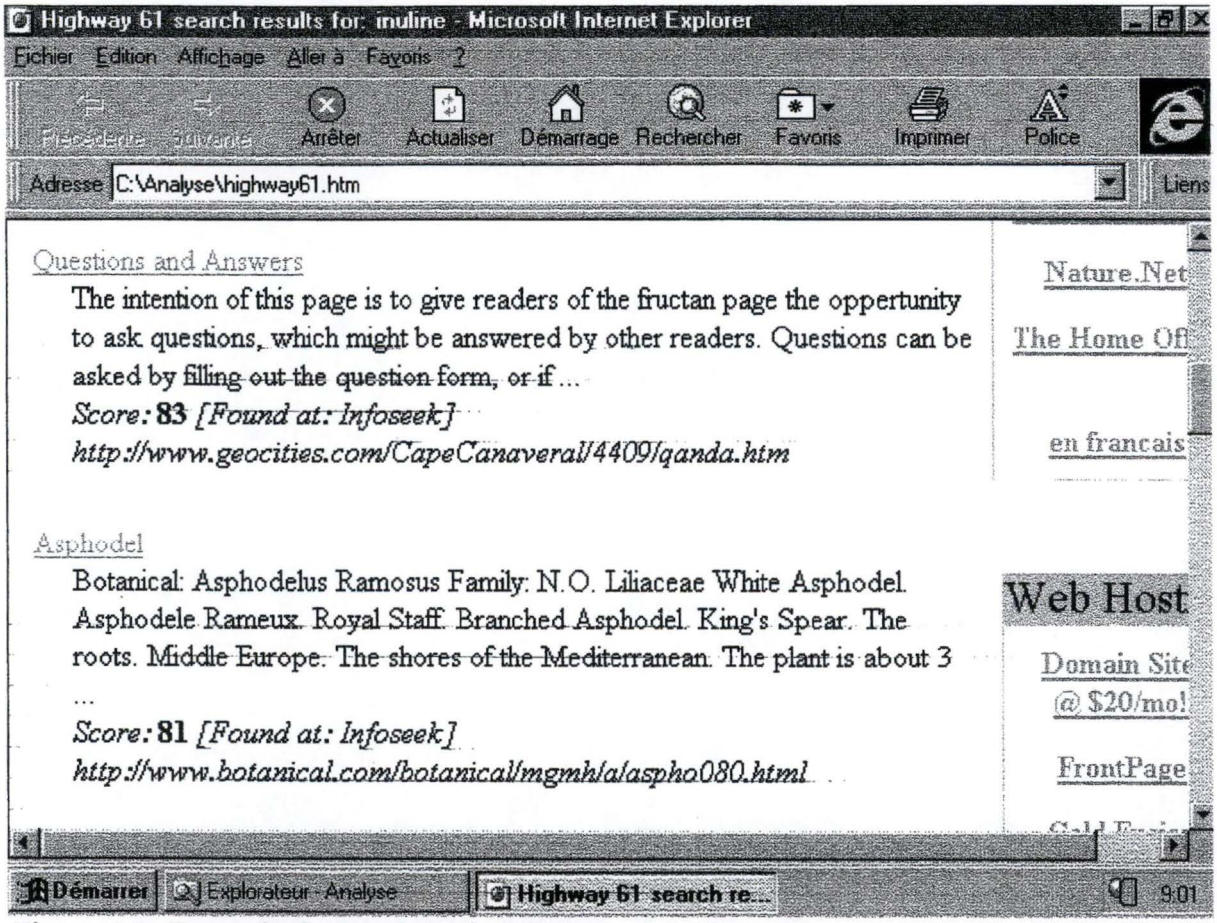


Figure 6.7 : Exemple des résultats de références trouvées par le méta-moteur de Highway61.com (mot clé "inuline")

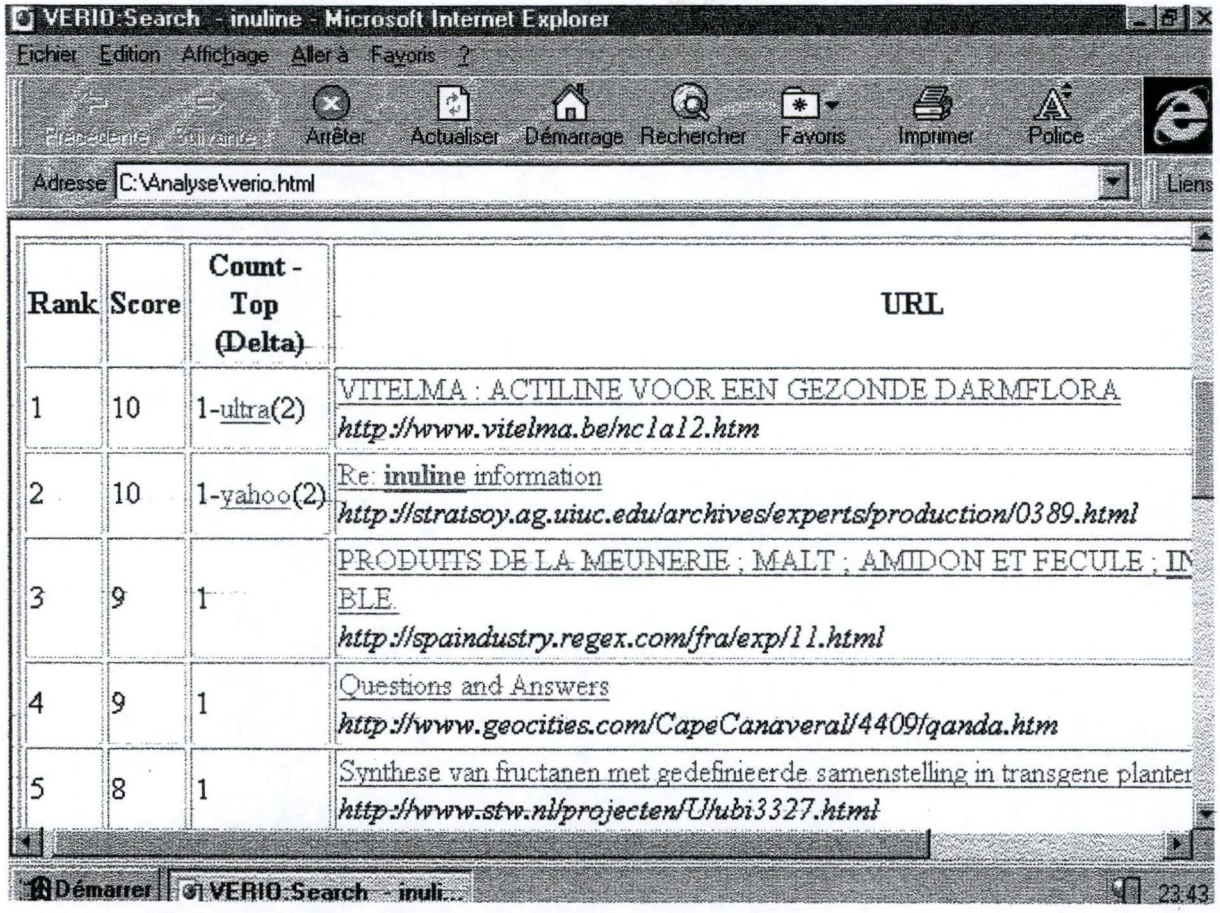


Figure 6.8 : Exemple des résultats de références trouvées par le méta-moteur de Verio.com (mot clé "inuline")

6.8. Traitements à réaliser

Comme il a été stipulé dans les buts du prototype à mettre au point (chapitre 5), les traitements à effectuer sont présentés brièvement dans l'ordre conceptuel, leur conception détaillée sera présentée par après:

- génération d'un formulaire pour le client;
- formulation d'une requête par le client;
- envoi d'une requête aux méta-moteurs;
- constitution d'une base de données des documents des références;
- calcul du poids des termes dans les documents;
- calcul de la similarité entre la requête et les documents;
- génération des documents pertinents;
- évaluation du client et *feedback.*

La figure 6.9 illustre le diagramme de flux de ces traitements, sur ce diagramme on y ajoute un traitement "Trouver Document Répondant" qui est effectué par les méta-moteurs.

A signaler que lors de la génération des documents pertinents, non seulement on effectue des mesures statistiques de correspondance entre le document et la requête, mais aussi le critère de la nouveauté d'un document est pris en compte.

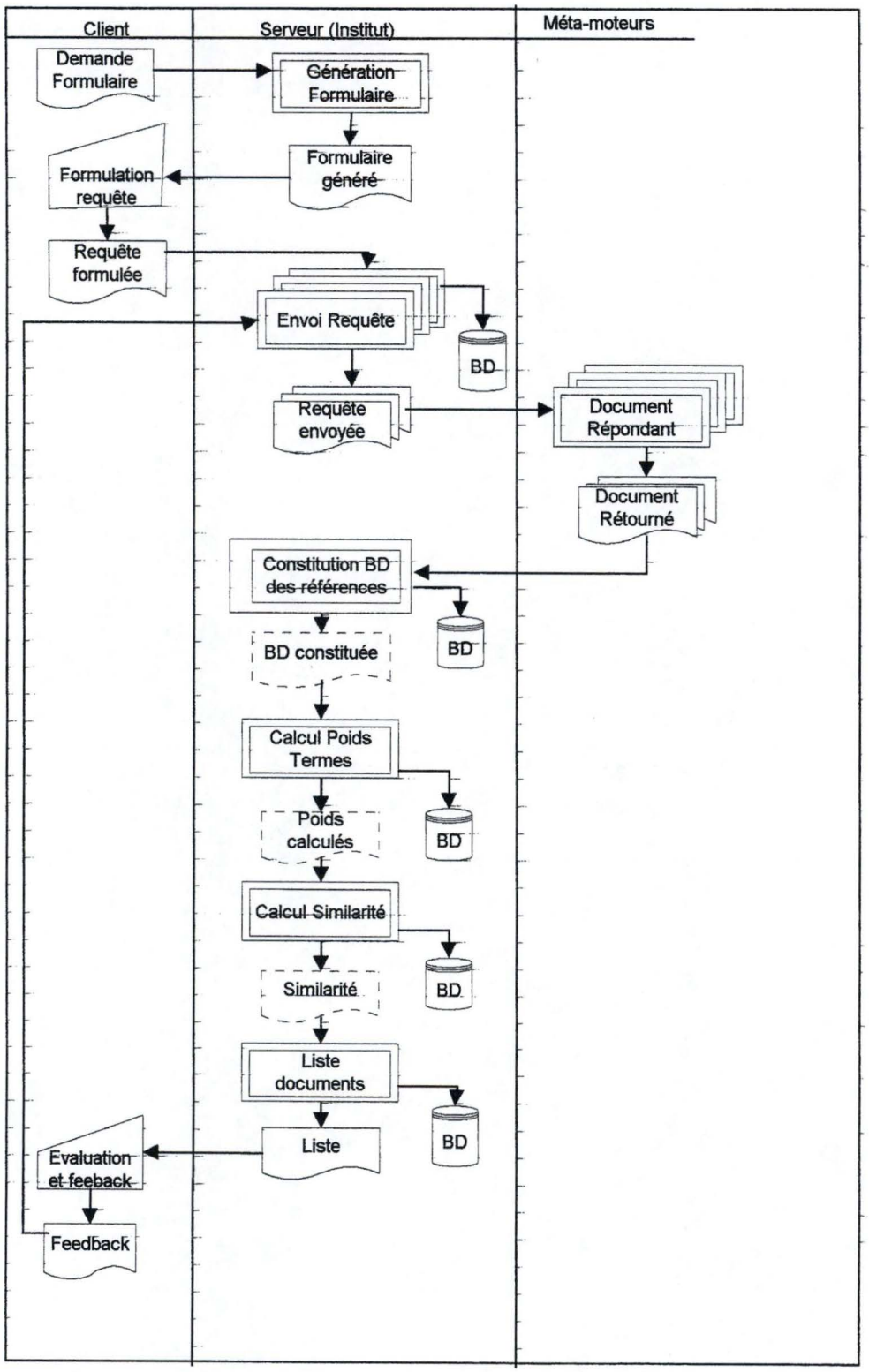


Figure 6.9 : Diagramme de flux des traitements à réaliser

6.8.1. Génération de formulaire

Comme on peut le constater sur le diagramme de flux de la figure 6.9, le client pour formuler sa requête doit d'abord se connecter au serveur de l'Institut. Ce dernier lui envoie un formulaire pour l'introduction des mots clés pour sa recherche.

6.8.2. Formulation de la requête

Pour ce traitement interactif, le client remplit le formulaire des mots clés et ce formulaire est soumis à l'application qui s'occupe d'aller interroger les différents méta-moteurs

6.8.3. Envoi de la requête au méta-moteurs

Ce traitement automatique est effectué par l'application se trouvant sur le serveur de l'Institut. La requête reçue du client est envoyée simultanément aux différents méta-moteurs, les documents répondant à la requête sont retournés à l'application par ces méta-moteurs. Ce traitement enregistre la requête dans la base de données.

6.8.4. Trouver Documents Répondant

Ce traitement est exécuté par les méta-moteurs interrogés, l'application se contente d'attendre les documents retournés pour continuer la suite des traitements.

6.8.5. Constitution de la base des données des références

Tous les documents retournés par les méta-moteurs vont être enregistrés dans une base de données. Pour la constitution de cette base de données, on va prendre l'URL du document et son titre (et dans la mesure du possible la date de sa mise sur le réseau Internet). Ainsi on va remplir une base de données dont le schéma conceptuel et physique sont illustrés respectivement par les figures 6.10 et 6.11.

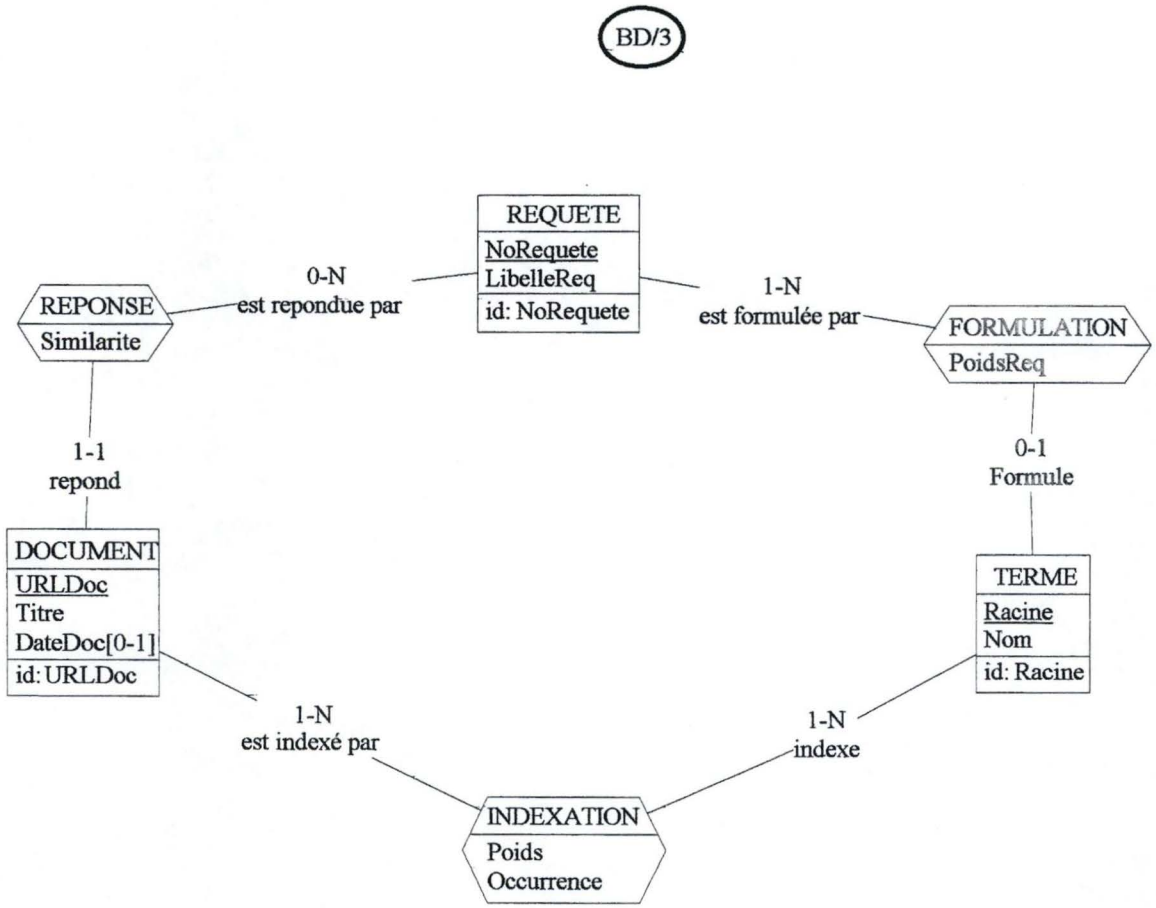


Figure 6.10 : Schéma conceptuel de la base de données

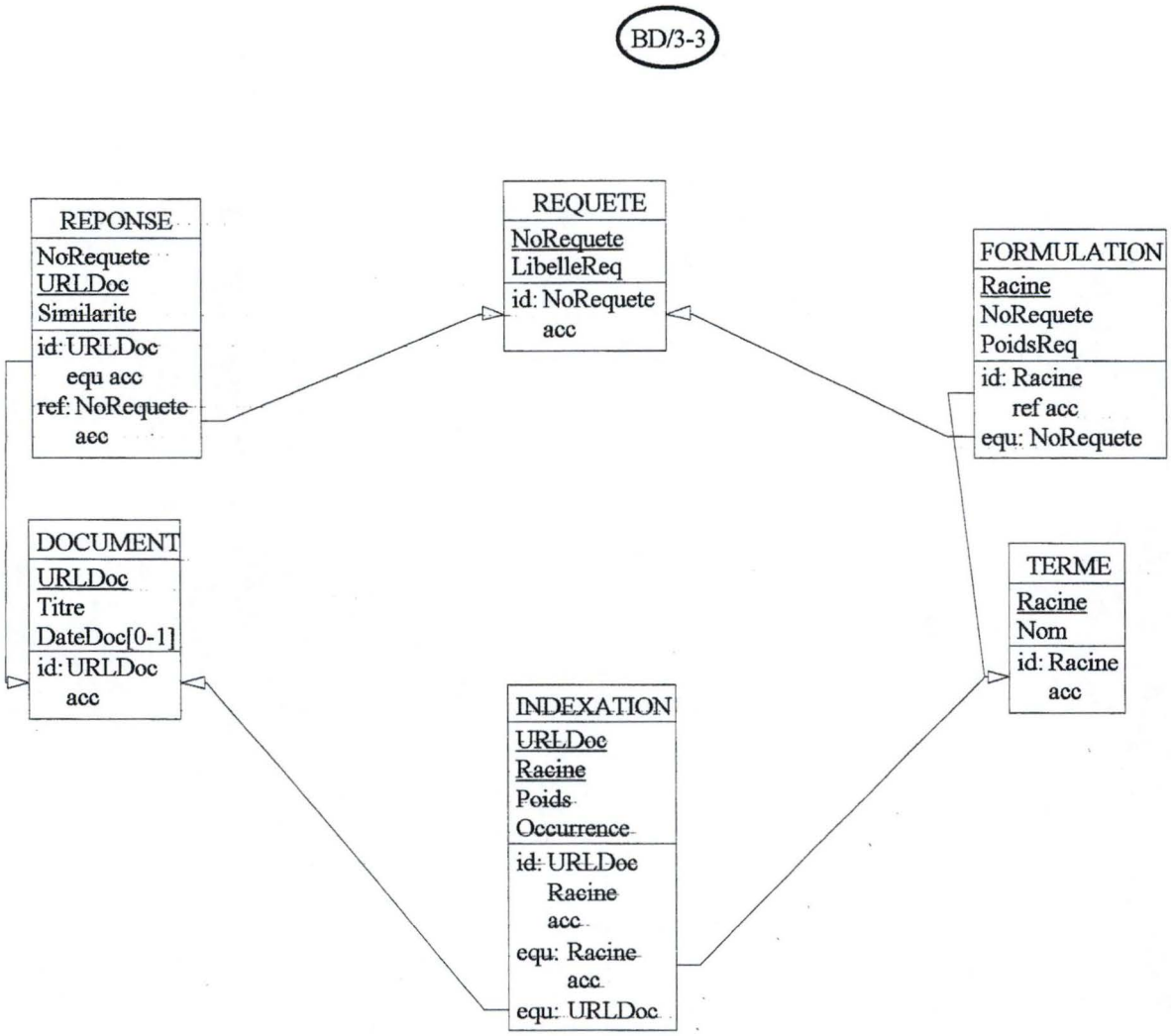


Figure 6.11: Schéma physique de la base de données

En regardant la figure 6.10, on peut constater que dans la base données, nous aurons :

- les types d'entité (T.E.) DOCUMENT, TERME, REQUETE;
- les types d'association (T.A) REPONSE, FORMULATION, INDEXATION;

Le T.A. REPONSE lie les T.E. DOCUMENT et REPONSE et ce T.A a comme attribut *Similarité*. En effet tout document se trouvant dans la base de données (BD), *répond* à une requête et entre le document et la requête il y a toujours une similarité calculée.

Le T.A. FORMULATION établit une association entre les T.E. REQUETE et TERME et dans le T.A. FORMULATION on y trouve un attribut *PoidsReq* qui indique le poids accordé à un terme dans une requête. Une requête est donc *formulée par* un à plusieurs termes. Et une requête est *répondue par* zéro (rôle) à plusieurs documents. Un terme qui se trouve dans la BD, *formule* zéro à une requête.

Le T.A. INDEXATION lie les T.E. DOCUMENT et TERME et les attributs *Poids* et *Occurrence* caractérisent ce T.A. L'attribut *Poids* indique la pondération accordée au terme dans le document tandis que l'attribut *Occurrence* indique le nombre d'occurrences du terme dans tout le document. Un document est *indexé par* au moins un terme et tout terme enregistré dans la BD *indexe* au moins un document.

Le T.E. REQUETE a comme attributs:

- *NoRequete* qui est un l'identifiant de ce T.E.;
- *LibelleReq* qui est une phrase descriptive de la requête (pour la sauvegarde de la requête);

Le T.E. TERME a comme attributs:

- *Racine* qui est la racine du terme après avoir enlevé les suffixes et l'attribut Racine constitue l'identifiant de ce T.E;
- *Nom* indique le terme avant l'enlèvement des suffixes.

Le T.E. DOCUMENT a comme attributs:

- *URLDoc* qui est l'endroit URL où est hébergé le document et cet attribut constitue l'identifiant de ce T.E;
- *Titre* qui est le titre du document;
- *DateDoc* qui est un identifiant facultatif pour indiquer la datation du document

La figure 6.11, illustre le schéma physique de la BD, le point important que l'on peut constater, c'est que la base de données va être constituée par six tables suivant:

- table REQUETE;
- table FORMULATION;
- table TERME;
- table INDEXATION;
- table DOCUMENT;
- table REPONSE.

Sans entrer plus en détail, on peut constater que chaque attribut identifiant constitue une clé d'accès (*acc* sur le schéma) pour notre base de données, et pour des raisons d'optimisation, certains attributs ne constituent pas une clé d'accès, par exemple dans la table INDEXATION, ULRDoc ne constitue pas une clé d'accès car on a une autre clé d'accès dans laquelle URLDoc constitue un préfixe, ...

Avec cette application, on va pouvoir :

- garder l'historique des requêtes;
- sauvegarder des requêtes et profils.

Garder l'historique des requêtes

D'après C. LELOUP (1997), lors d'une session de recherche, la possibilité de revenir à une requête précédente est très appréciable, en particulier pour exploiter de nouveau une requête précédemment traitée. C'est ce qu'on appelle historique, le logiciel mémorisant les requêtes successives ainsi que le nombre de réponses et les clés des documents. Notre requête va être identifiée par le numéro, qui s'incrémente à chaque nouvelle question, la formulation de la requête, telle qu'elle a été adressée est conservée dans la base de données.

Sauvegarde de requêtes et profils

La sauvegarde de requête permet à l'utilisateur de donner un nom à sa recherche, quelque chose plus explicite qu'une équation compliquée, pour pouvoir, d'une

session à l'autre, la lancer au besoin (C. LELOUP, 1997). Dans notre système, on va effectuer la sauvegarde d'une requête par l'attribut *LibelleReq* de la table REQUETE.

La notion de profil est une extension de la sauvegarde de requêtes. L'utilisateur définit une requête (ou plusieurs) qui correspond à son sujet d'intérêt. Si les documents pertinents sont trouvés qui n'étaient pas présents lors de la précédente, l'utilisateur est avertie par messagerie ou lorsqu'il se connecte à la base de l'information (C. LELOUP, 1997).

Notre système va avertir de l'arrivée de nouveaux documents pertinents par rapport à une requête donnée lorsque l'utilisateur se connecte au système d'information.

6.8.6. Calcul des poids de termes dans un document

Le raisonnement se rencontre chez Salton (1983) et chez D'haeyère (1995), une passe est effectuée dans la collection des documents. Les termes caractérisant les documents sont conservés dans les vecteurs des mots clés des documents. Ensuite un calcul est effectué pour déterminer le poids des mots en fonction du nombre d'occurrences dans le document et dans la collection des documents.

La fonction retenue pour calculer le poids des mots dans un document est :

$$W_{ik} = \frac{tf_{ik} * \log\left(\frac{N}{n_k}\right)}{\sqrt{\sum_{j=1}^{t} (tf_{ij})^2 * \left[\log\left(\frac{N}{n_j}\right)\right]^2}} \qquad (6.1)$$

Où w_{ik} est le poids du terme T_k dans le document D_i

tf_{ik} est la fréquences d'occurrences du terme T_k dans le document D_i

N est le nombre de documents dans le collection

n_k est le nombre de documents comprenant le terme T_k

t est le nombre de termes distinctes dans la collection de documents

6.8.7. Calcul de la similarité entre requête et les documents

Pour calculer la similarité entre une requête et un document , la fonction suivante est utilisée:

$$Sim(R,D) = \sum_{k=1}^{t} w_{rk} * w_{dk} \qquad (6.2)$$

Où R est la requête

D est le document

w_{rk} le poids du terme T_k dans le *représentant de requête R*

w_{dk} le poids du terme T_k dans le *représentant de document D.*

Cette fonction retourne une valeur entre 0 et 1 selon le degré de similarité. Plus la similarité entre une requête et un document est proche de 1 plus ce document est similaire à la requête (Salton ,1988; D'haeyère, 1995).

6.8.8. Détermination des documents pertinents et générations des nouveaux termes pour le *feedback*

Ce traitement se base sur la similarité entre les documents et la requête pour effectuer un tri sur la similarité. Les documents dont la similarité avec la requête est plus élevée vont être proposés au client avec la pertinence (l'expert du domaine va intervenir pour déterminer le seuil de similarité pour dire si un document est pertinent ou pas).

Les termes les plus fréquents dans les documents pertinents et qui ne sont pas présents dans la requête vont être proposés au client pour améliorer sa requête

(Crestani, 1995) dans la suite on pourrait comparer avec la formule de *relevance feedback* de Salton (équation numéro 3.20).

6.8.9. Evaluation du client et feedback

Le client évalue les résultats (voir figure 6.9) obtenus par l'application et le système lui propose des mots clés à ajouter ou à retirer de la requête pour affiner sa recherche, et le processus de recherche recommence (mécanisme de *feedback).*

Chapitre 7 :

7. Implémentation

7.1. Introduction

Le langage Java, comme on l'a déjà vu, est un langage qui a été mis au point pour répondre aux besoins de la programmation Internet. Ce langage offre des fonctionnalités pour effectuer une interaction entre le client et le serveur. Par exemple API Servlet Java vue au point 6.6.2.

Par ailleurs, le langage Java permet la programmation des *threads*. Dans ce travail, on va exploiter les mécanismes de "*thread*" et c'est pourquoi, il semble utile de donner sa définition que l'on trouve chez E. Puybaret (1998), et la raison pour laquelle on privilégie ce mécanisme.

L'environnement de la Machine Virtuelle Java est *multi-threads*. L'équivalent de *thread* pourrait être *tâche* en français, mais pour éviter la confusion avec la notion de système *multitâches*, nous emploierons le mot *thread* plutôt que *tâche*. Le fait que Java permettent de faire tourner en parallèle plusieurs *threads* lui donne beaucoup d'intérêt : ceci permet par exemple de lancer le chargement d'une image sur le Web (ce qui peut prendre du temps), sans bloquer votre programme qui peut ainsi effectuer d'autres opérations.

L'application que l'on va réaliser utilise une base de données Oracle, l'API JDBC (Java DataBase Connection) permet d'interactions avec des bases des données, et des requêtes SQL sont effectuées grâce à cette API .

Evidemment, ceci va sans commentaires, avec le langage Java, on exploite la programmation orientée-objet où chaque objet est caractérisé par ses attributs et ses méthodes.

7.2. Classe Node

Buts : Cette classe va être utilisée par la liste chaînée.

Attributs :

- **private Object elem** (qui est l'élément de la liste chaînée);
- **private Node** (reference au nœud suivant);

Méthodes :

- **Node() :** constructeur;
- **Node (Object elem) :** constructeur qui affecte un élément au nœud de la liste chaînée
- **Node (Object elem, Node n) :** constructeur qui affecte un élément à un nœud et qui fait la référence au nœud suivant;
- **GetElem() :** retourne un objet;
- **Void setNextElem(Node NewNode) :** fait la référence au nœud suivant.

7.3. Classe objectNotFoundException

Buts : Cette classe va être utilisée par la classe Linklist pour générer l'exception

ObjectNotFoundException() : constructeur.

7.4. Classe Linklist

Buts : Cette classe permet de travailler avec une liste d'objets génériques et elle utilise la classe Node (Cohen, 1999)

Attributs :

- **private Node Head** (référence au premier Nœud de la liste) ;

- **private Node Cursor** (référence au Nœud courant).

Méthodes :

- **Linklist() :** constructeur;
- **Boolean isEmpty() :** teste si la liste est vide;
- **Void resetList() :** efface la liste;
- **endOfList() :** teste si on est à la fin de la liste;
- **Void goToNext():** permet d'avancer au nœud suivant ;
- **Node getNextElem() :** retourne l'élément suivant;
- **String toString() :** retourne la valeur du nœud sous forme d'un string .

7.5. Classe Arbre

Buts : Pour augmenter l'efficacité du système, D'HAEYERE (1995, p101) proposé d'utiliser un arbre binaire générique d'objets . Et cet arbre est trié et l'arbre va utiliser la classe "ObjcetNotFoundException" au cas d'échec

Les méthodes utilisées :

- **Arbre() :** constructeur qui initialise l'arbre vide;
- **Void Add(Object elem) :** ajout dans l'arbre l'élément passé en paramètre ;
- **getElem(Object elem) :** recherche dans l'arbre l'élément passé en paramètre ;
- **Void DeleteAllInfo() :** détruit tous les éléments de l'arbre;

7.6. Classe Requete

Buts : cette classe reçoit une requête du client à partir du formulaire HTML, cette classe va garder l'adresse du client, la requête reçue du client, la liste des racines des mots clés caractérisant cette requête et l'arbre de ces racines

Méthodes utilisées :

- **Requete() :** constructeur d'une requête vide;
- **Requête(String url, DocumentManager manager, String requete) :** constructeur d'une requête associé à l'adresse du client, lié à un DocumentManager et portant la phrase de la requête passée en paramètre. Une liste et un arbre vide sont créés.
- **Void UpdateTerm () :** par cette méthode on va effectuer la mise à jour dans la base des données des informations concernant les termes qui caractérisent une requête;
- **RetrieveTerm() :** cette méthode permet la recherche des termes caractérisant la requête depuis la base des données;
- **Void SetRepresentant() :** cette méthode crée le *Représentant d'une requête* à partir des termes caractérisant la requête obtenus . En effet par cette méthode, on va parcourir la phrase associée mot par mot. Si le mot est présent dans le vecteur des mots fréquents *(Stopwords)* du DocumentManager, alors il est ignoré. Sinon, le mot est réduit à sa racine et cette racine est conservé dans la liste et l'arbre des racine de la requête;
- **GetRepresentant() :** cette méthode renvoie le *Représentant d'une requête;*
- **CalculateWeight() :** calcule le poids de tous les racines de la requête .

7.7. Classe EnvoiRequete

Buts : cette classe implémente l'interface Runnable (en langage Java) afin de pouvoir utiliser les threads. Dans cette classe on y trouve des méthodes qui permettent d'envoyer simultanément des requêtes à plusieurs méta-moteurs.

Attributs de la classe :

Thread Profusion;

Thread Mamma;

Thread Cyber411;

Thread Hayway61;

Thread Infind;

Thread verio;

Thread Crawler;

Méthodes utilisées :

- **EnvoiRequete() :** constructeur; (c'est ici que l'on doit paramétrer tous les caractéristiques de chaque méta-moteur;
- **Void start () :** avec cette méthode, on démarre chaque thread;
- **Void stop () :** pour l'arrêt du thread;
- **Void run() :** ici, on va effectuer l'envoi de la requête à chaque méta-moteurs. Les threads vont écrire les résultats de leur recherche dans un fichier unique"REPONSE".

7.8. Classe ChercheReference

Buts : cette classe s'inspire du travail réalisé par BALDI (1997). A la page 53 de son travail, on peut lire "*avant toute chose, un thread est crée afin d'accélerer la recherche et de pouvoir la stopper quand on le désire. L'application se connecte alors à la page WEB dont l'utilisateur a donné l'adresse (si cette adresse est valide. Sinon, un message d'erreur est renvoyé). L'application crée alors un nouveau flux de données (InputStream dans le langage JAVA), afin de pouvoir lire le contenu da la page WEB. Une fois cela fait, la recherche commence au sein de cette page WEB: tant qu'on n'est pas arrivé à la fin du fichier HTML, on vérifie ligne par ligne la présence de l'identificateur de lien href = "http://. Si on trouve un tel identificateur, on copie ce qui vient après dans un tampon.*

Si on trouve href="http://www.nom.suffixe./page.html", on place www.nom.suffixe/page.html dans le vecteur ou bien si on trouve href="http://www.nom.suffixe./page.htm, on va aussi prendre ce lien dans le vecteur.

Dans notre travail, l'application, au lieu de se connecter à une page WEB, l'application se connecte au fichier "REPONSE", créé par la classe EnvoiRequete. On va parcourir ce fichier et toutes les références se trouvant dans ce fichier sont mis dans un vecteur . On se rappelle que les références sont toujours les URLs qui indiquent l'endroit où on va trouver le document.

Méthodes utilisées :

- **Void start() :** pour démarrer le multithreading;
- **Void stop() :** pour arrêt du thread
- **Void run() :** pour rechercher toutes les références contenues dans le fichier REPONSE et les mettre dans un Vecteur;
- **supprimedoublon() :** pour la suppression des URLs dans le vecteur qui se répètent. En effet, si les URLs se répètent, cela signifie que l'on va avoir plusieurs fois le même document.
- **Gettitle() :** pour chaque document trouvé on va garder son titre. On va pouvoir récupérer le titre du document grâce aux balises <TITLE> </TITLE> du langage HTML. Les instructions se trouvant entre ces deux balises constituent le titre du document. C'est ce titre qui sera proposé au client et ce titre sera enregistré dans la base des données;
- **GetDocument() :** cette méthode va sur un URL et télecharge le document. (Ici, on se demande s'il n'y a pas moyen de déterminer le représentant de document sans télécharger le document, ce qui peut augmenter les performances du système).

7.9. Classe Racine

Le même raisonnement pour cette classe se trouve dans le travail de D'HAEYERE (1995, p.98) avec quelques modifications. Elle est chargée de gérer les informations pour cette racine, telles que le nombre d'occurrences et la pondération de cette racine.

Les méthodes utilisées :

- **Racine() :** constructeur pour une racine vide;
- **Racine(String term) :** constructeur de racine copiant la chaîne de caractères donnée en paramètre. Le nombre d'occurrences est mis à 1 et le poids à 0;
- **Void afficherRacine()** : pour afficher l'instance de la racine.

7.10. Classe DMRacine

Cette classe est prise dans le travail de D'HAEYERE (1995, p.98-99) avec quelques modifications. Elle gère des informations concernant des racines, mais pour la classe DocumentManager. Elle conserve le nombre de documents dans la collection possédant cette racine ainsi que le nombre d'occurrences de cette racine dans la collection. Le dernier document dans lequel cette racine a été localisé est conservé par cette classe.

Méthodes utilisées:

- **DMRacine**(): constructeur d'une chaîne de caractères vide;
- **DMRacine(String DMterm)** : constructeur d'une chaîne de caractères en copiant la chaîne passée en paramètre. Cette méthode initialise le nombre d'occurrences à 1, et met à -1 le numéro du dernier document dans lequel elle a été localisée et met à 0 le nombre de documents dans lesquels cette racine est présente;
- **AfficherDMRacine() :** pour l'affichage d'une instance de la classe DMRacine.

7.11. Classe Document

En regardant la base de donnée à la figure 6.8, et en se basant sur le travail de D'HAEYERE (1995, p. 102 et p.103), on constate qu'il y a une table DOCUMENT. Cette classe va donc permettre d'aller écrire des caractéristiques du document dans la base de données.

Attributs :

- URLDoc;
- TitleDoc;
- Date;
- liste de racines des mots;
- Arbre binaire des racines des mots;
- Requête ;
- Similarite;

Méthodes utilisées :

- **Document()** :constructeur d'un document vide, sans fichier associé, sans titre, avec une liste et un arbre vide de racines;
- **Document(String url, DocumentManager manager, String titre) :** constructeur d'un document associé à un fichier donné, lié à un DocumentManager et portant le titre passé en paramètre. Une liste et un arbre vide sont créés;
- **Void UpdateTerm () :** par cette méthode on va effectuer la mise a jour dans la base des données des informations concernant les termes qui caractérisent le document;
- **RetrieveTerm() :** cette méthode permet la recherche des termes caractérisant le document depuis la base de données;
- **SetRepresentant() :** cette méthode crée le *Représentant d'un document* à partir des termes caractérisant le document. En effet par cette méthode, on va parcourir le fichier associé mot par mot. Si le mot est présent dans le vecteur des mots fréquents *(Stopwords)* du DocumentManager, alors il est ignoré. Sinon, le mot est réduit à sa

racine et cette racine est conservé dans la liste et l'arbre des racinesdu document. Les informations sur les racines dans le DocumentManager sont modifiées de manière appropriée.

- **GetRepresentant() :** cette méthode renvoie le *Représentant d'un document;*
- **CalculateWeight() :** calcule le poids de tous les racines du document et enregistre les poids dans la base de données;
- **UpDateRequete () :** la mise à jour de la requête dans la base de données;
- **RetrieveRequete() :** permet d'aller rechercher dans la base de donnée la requête à laquelle le document répond.

7.12. Classe DocumentManager

Buts : Cette classe prise chez D'HAEYERE (1995, p.103) avec quelques modifications permet la facilité de traitement de tous les documents.

Attributs de la classe :

- liste de *Stopwords*;
- liste des racines présentes dans la collection;
- arbre binaire des racines présentes dans la collection;

Méthodes utilisées :

- **DocumentManager() :** constructeur;
- **Add () :** ajout du document passé en paramètre aux documents qu'elle prend en gestion;
- **DeleteDocuments() :** détruit les documents dont il prend en gestion;
- **SetTotalTerm() :** parcourt toute la collection de documents et garde tous les termes présents dans la collection dans une liste (ou arbre binaire);
- **GetTotalTerm() :** renvoie la liste de tous les termes distincts de la collection de document;

- **GetTotalDocument():** renvoie le nombre total de documents dans la collection ;
- **GetNombreDocTerm (Racine term) :** cette méthode renvoie le nombre de documents comprenant le terme;
- **CalculateWeight() :** parcourt la collection des documents et calcule le poids de tous les termes;
- **EliminateTerm() :** élimine les termes qui ne sont pas pertinents;
- **ReadStopWords(String fichier) :** construit l'arbre des mots fréquents à partir du fichier passé en paramètre.

7.13. Classe Similarity

Buts : Par cette classe, on va calculer la similarité de la requête (son *Représentant)* avec le document (le *Représentant de document*). Les documents dont la similarité avec la requête est élevée vont être proposés aux client.

Méthodes utilisées :

- **Simirarity() :** constructeur;
- **CalculateSimil(Document doc, Requete requ) :** calcule la similarité entre une requête et un document passé en paramètre;
- **GetSimil() :** renvoie la similarité du document avec la requête.

7.14. Classe ReponsetoClient

Buts : Les documents dont la similarité avec la requête est élevée vont être adressés au client ; les premiers termes se trouvant dans les documents pertinents et qui ne se trouvent pas dans la requête sont proposés au client pour améliorer sa requête, et le processus de recherche recommence.

Méthodes utilisées :

- **ReponsetoClient(String adresse) :** constructeur qui initialise l'adresse du client qui a formulé la requête;

- **SelectPertinent() :** Avec cette méthode, on va interroger notre base données sur le critère de similarité. L'expert du domaine va intervenir pour juger si tel document est pertinent pour lui; ainsi on va déterminer le seuil de similarité à partir duquel le document est considéré pertinent à la requête;

- **GetNewTerm() :** cette méthode permet de proposer des nouveaux termes au client pour affiner sa requête.

7.15. Classe Porter

Cette classe permet d'obtenir une racine d'un mot en enlevant les suffixes, cette classe met en œuvre l'algorithme de Porter et elle va être implémentée telle qu'elle a été définie par J. Keyes (janvier, 1998).

7.16. Classe vue_BD

Buts : cette méthode permet la manipulation de la bases de données, au lieu de travailler tout de suite sur la base des données on va travailler avec les vues et on enregistre dans la base de données après.

Méthodes de la classe :

- **vue_BD () :** constructeur;

- **void AddDoc(String url, String title, Date date) :** cette méthode ajoute un document dans la table DOCUMENT, INDEXATION et REPONSE;

- **getDocument(String url) :** recherche si un document se trouve dans la base de données et l'envoie se document;

- **getAllDocument()** : envoie tous les documents se trouvant dans la base de données ;

- **void AddTerm(String racine, String nom, String urldoc, integer occurrence float poids)** : cette méthode permet l'ajout des termes dans la table TERME et INDEXATION;

- **getTerm (String racine) :** envoie le terme passé en paramétre;

- **void UpDateTerm (String racine, String nom, String urldoc, integer occurrence float poids) :** la mise à jour des termes;

- **void AddSimilarity(String url, integer norequete, float similarite) :** ajout de la similarité de la requête avec le document dans la base des données;

- **GetSimilariry(String url, norequete) :** envoie la similarité d'un document avec une requête;

- **GetOccurrence(String url String terme)** : renvoie l'occurrence d'un terme dans un document;

- **GetPoids(String url String terme)** : renvoie le poids d'un terme dans un document;

- **void AddRequete(Integer Norequete, String libbellereq) :** enregistre le libellé d'une requête;

- **void AddFormulation (Integer norequete, String racine, float PoidsRequete)** : enregistre le numéro de la requête, ses racines et le poids accordé à chaque racine ;

- **GetRequete(Integer norequête) :** envoie le numéro de la requête, son libellé, et les termes qui formulent la requête ;

- **GetAllRequete()** : retourne toutes les requêtes enregistrées dans la base de données.

7.17. Classe BD

Buts : cette classe s'occupe de l'ouverture de la base de données, de la fermeture de la base de données avec la vérification des contraintes d'intégrités de cette base de données.

Méthodes :

- **void Ouvrir_BD ()** : ouvre la base de données ;
- **void Fermer_BD()** : ferme la base de données ;
- **void Begin_transaction() :** ouverture d'une transaction ;
- **void Comit_Tansaction() :** clôture avec confirmation;
- **void Abort_Transaction() :** clôture avec annulation.

7.18. Classe Coordinator

Buts : Dans cette classe, on va définir le programme principal qui génère un formulaire HTML pour le client afin qu'il puisse formuler sa première requête et dans cette classe que la base de données est ouverte, c'est dans cette classe que la servlet est initialisée.

Méthodes :

- **Coordinator() :** constructeur; pour initialisation de la servlet et la connexion à la base de données.
- **GenereFormulaire() :** génération du formulaire qui permet la formulation de la requête par le client .

Chapitre 8 :

8. Conclusion et perspectives

Dans ce mémoire, les techniques de recherche d'information pour améliorer la qualité des références ont été traitées. Les méthodes d'indexation, les méthodes de calcul de la similarité entre une requête et les documents et les méthodes de formulation de requêtes ont été abordées.

Les nouvelles technologies pour la programmation de l'Internet et leurs apports pour les systèmes de recherche d'information ont été mentionnées. La programmation des scripts CGI, le langage Java et HTML étant conçus pour s'adapter au World Wide Web.

L'analyse et la spécification d'une application permettant l'amélioration de la qualité des références fournies par les moteurs de recherche sur le Web ont été réalisées. Cet outil calcule la similarité entre une requête et des documents en se basant sur des indices statistiques de la présence des mots clés dans les documents et sur des pondérations adaptées grâce à l'information fournie par le spécialiste du domaine.

Grâce à ce programme, un utilisateur pourra choisir les mots clés de son sujet et il sera à mesure, non seulement d'obtenir des documents pertinents répondant à sa requête, mais aussi il pourra suivre, d'une manière automatique, l'évolution de l'apparition de nouveaux documents sur le réseau Internet.

L'application spécifiée proposera à un utilisateur des nouveaux mots clés pour améliorer sa recherche.

Dans la suite de ce travail, nous envisageons l'implémentation du prototype dans le domaine de la sucrerie.

Dans l'avenir, les recherches pourront continuer avec l'utilisation de thesaurus pour affiner la requête et les techniques d'intelligence artificielle comme des réseaux de neurone pour la recherche d'information pourront être envisagées.

Index bibliographique

ANUP K. GHOSH [1998]. *E-Commerce Security Weak Links Best Defenses*, John Willey & Sons, New York, U.S.A, 1998.

BALDI J. [1997*]. Reconnaissance automatique des liens dans une page web : identification et représentation*. Mémoire de fin d'études, Institut d'Informatique, FUNDP, Namur, Belgique, 1997.

BÄRTSCHI M.A. [1984]. *Term Dependance in Information Retrieval Models.* A dissertation submitted to the Suiss Federal Institute of Technology (ETH) Zürich for the degree of Doctor of Technical Sciences, 1984.

CHALEAT P., CHRNAY D., FLUCKIGER F.[1997]. *HTML et la programmation des serveurs Web*, Eyrolles, Paris, 1997.

COHEN C. [1999]: *Linklist.* http://www.cs.binghamton.edu/~ccohen/

CRESTANI Fabio [1995] *Implementation and evaluation of a relevance feedback device based on neural networks*,Dipartimento di Electronica e Informatica, Universita' di Padova, , dans http://www.dcs.gla.ac.uk/ir/papers/

DELONNOY F [1998], *Servlets*, Htttp://jserv.javasoft.com/products/java-server/documentation/webserver.

D'HAEYERE V. [1995], *L'application des mécanismes d'information retrieval pour la construction automatique de systèmes hypertextes.* Mémoire de fin d'études, Institut d'Informatique, FUNDP, Namur, Belgique, Septembre, 1995.

DINANT J. M. [1999], *Les traitements invisibles sur l'Internet*. Séminaire au cours de la Sécurité et la Fiabilité des Systèmes d'information, 2ème Licence et 3ème Maîtrise Informatique, FUNDP, Namur, Belgique, 1999.

GIRARDI GUTIERREZ M.R.[1995], *Classification and Retrieval of Software through their Descriptions in Natural Langage*. Thèse de doctorat, Université de Génève, 1995.

GOFFINET L. [1999], *Servlets,* Communication interne, Institut d'Informatique, FUNDP, Namur, Belgique, 1999.

GOFFINET L., NOIRHOMME-FRAIRURE M. [1996], *Automatic Hypertext Link Generation Based on Similarity Measures Between Documents,* Institut d'Informatique, FUNDP Namur, Belgique, 1996

LELOUP C. [1997], *Moteurs d'indexation et de recherche. Environnement client - serveur, Internet et Intranet*, Eyrolles, Paris, 1997.

KEYES J.[1998], *PorterStemmer.java,* http://ruby.ils.unc.edu/keyes/java/porter/index.html

PUYBARET E. [1998], *Cours Java,* http://eteks.com/coursjava/

SALTON G. BUCKLEY C. [1988]. *Improving retrieval Performance by Relevance Feedback*. Departement of Computer Science Cornell University, Ithaca, NY 14853-7501,U.S.A., February, 1988.

SALTON G., MCGILL M.J. [1983] *Introduction to Modern Information Retrieval*, McGraw-Hill, New York, U.S.A, 1983.

VAN RIJSBERGEN [1979]. *Information retrieval.* Departement of Computing Science, University of Glasgow, dans http://www.dcs.gla.ac.uk/ir/papers/

Errata

p.11§4 ligne 4	Lire " …document (*FREQik*) ou la fréquence … " au lieu de " ...document (*FREQik* ou la fréquence… "
p.20 équation 3.12	Lire " *i* " à la place de " *k* "
p.26 §2 ligne 4	Lire "…soustrayant des termes pris…" au lieu de "… soustrayant de termes des termes pris…"
p.31 Tableau 4.1 ligne 3 colonne 2	A supprimer " Annexe I "
p.53 §1 ligne 1	Lire "… PATH_INFO =/ceci/est/la/suite? … " au lieu de "…PATH_INFO =/ceci/est/la/fin … "
p.55 §3 ligne 1 p.55 §4 ligne 4 p.56 §2 ligne 1 p.56 §3 ligne 1 p.57 §1 ligne 1 p.85 §7 ligne 1	Lire " Delannoy " au lieu de " Delonnoy "
p.57 §2 ligne 4	Lire " (F.Delannoy, 1998) " au lieu de " (F.Delannoy, 1988) "
p.85 §3 ligne 1	Lire "… CHARNAY …" au lieu de "…CHRNAY…"

Légende :

p.= page

§ = paragraphe

ANNEXES

ANNEXE I

Voici la liste de quelques méthodes permettant d'obtenir des informations passées par le formulaire HTML. Cette liste est tirée du *package HttpServletRequest.* Cette package se définit ainsi:

Interface

javax.servlet.http.HttpServletRequest

public interface **HttpServletRequest** extends **ServletRequest**

Cette interface qui permet l'utilisation du protocole HTTP, retourne les données transmises par le client au servlet qui se trouvant sur le serveur.

Les méthodes

- **getCookies**

public abstarct Cookies [] getCookies()

Cette méthodes retourne un tableaux de cookies retrouvés dans une requête.

- **getMethod**

public abstarct String getMethod()

Cette méthode retourne la méthode HTTP (par exemple GET, POST) avec laquele la requête a été formulée. La valeur retournée est similaire à la variable d'environnement REQUEST_METHOD pour les CGI.

- **getServletPath**

public abstarct String getServletPath()

Cette méthode envoie une valeur équivalente à celle de la variable d'environnement SCRIPT_NAME pour les CGI.

- **getPathInfo**

public abstarct String getPathInfo()

Par cette méthode, on obtient une valeur analogue à la variable d'environnement PATH_INFO pour les CGI.

- **getPathTranslated**

public abstarct String getPathTranslated()

Cette méthode retourne une valeur analogue à la variable d'environnement PATH_TRANSLATED pour les CGI.

- **getQueryString**

public abstarct String getQueryString()

Cette méthode retourne une valeur comparable à celle de la variable d'environnement QUERY_STRING pour les scripts CGI.

- **getRemoteUser**

public abstarct String getRemoteUser()

Cette méthode retourne le nom de l'utilisateur qui a formulée la requête quand le système le permet. La valeur retournée est semblable à celle de la variable d'environnement REMOTE_USER pour les scripts CGI.

- **getAuthType**

public abstarct String getAuthType()

Cette méthode retourne le schéma d'authentification de la requête. La variable similaire pour les scripts CGI est la variable AUTH_TYPE.

D'autres méthodes existent dans cette interface **HttpServletRequest.** Par exemples les méthodes permettant de récupérer l'en-tête de la requête, la date de la requête se trouvant dans l'en-tête de cette requête, l'identification de la requête, de la session ...

L'*API Servlets Java* ne permet seulement de recevoir des données à partir du client, elle permet aussi d'envoyer des données au client par les méthode de l'interface **HttpServletResponse.**

L'*API Servlets Java* permet aussi d'interroger des bases des données comme Oracle, .. via des *drivers* qui permettant la connexion aux bases de données.