4

Construction des index

4.1 Approche de la construction d'index

Les services web d'indexation présentés précédemment doivent exploiter et rechercher dans des index physiques qui ont été créés suite à une indexation. Dans ce que suit nous présentons l'approche de l'indexation suivie dans ce travail, c'est à dire les étapes de la construction des index.

L'indexation est une forme de reformulation. Indexer, c'est reformuler le contenu d'un document dans une forme plus adaptée à son contexte d'exploitation dans une application donnée, on indexe donc, en vue d'une application. On ne parle plus seulement d'indexation mais également d'enrichissement, d'annotation, de marquage et de métadonnées [23].

Nous avons au départ un ensemble de ressources (fichiers) hétérogènes. Selon le type

de la ressource, deux traitements sont possibles (Figure 4.1) :

1. Indexation automatique : liée aux documents textuels, ex : fichiers textes.
2. Indexation manuelle : ça concerne les annotations faites par des humains (e.g. médecins, etc.) sur les ressources non textuelles, e.g. une vidéo, une image.

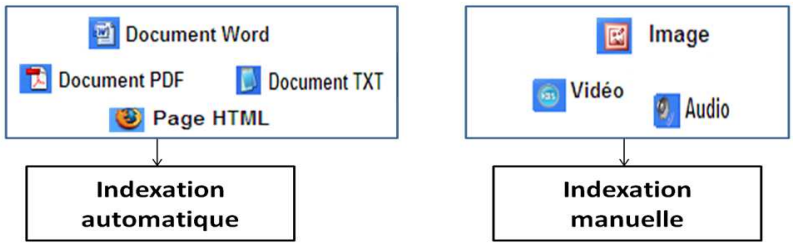

Fig. 4.1: Types d'indexation

4.1.1 Indexation automatique

Dans ce type d'indexation, nous nous inspirons des travaux de Baziz et al. [24][25][26]. La construction des index utilisés par les services web proposés dans l'indexation automatique, se fait en deux phases principales : l'indexation syntaxique et l'indexation sémantique, comme il est montré dans la Figure 4.2.

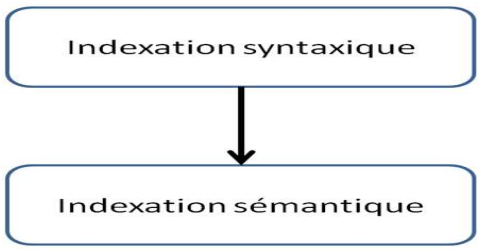

Fig. 4.2: Phases de l'indexation automatique

a- Indexation syntaxique

L'objectif de l'indexation syntaxique est d'extraire les termes des documents et de les stocker avec leur nombre d'occurrences dans un index physique (voir la Figure 4.3).

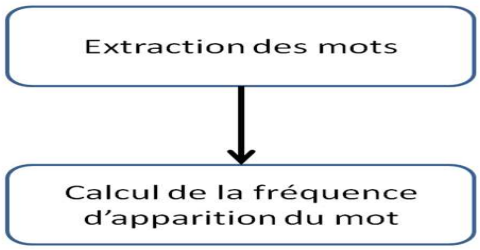

Fig. 4.3: Phases de l'indexation syntaxique

- Extraction des mots

Dans cette première phase de construction des index, il s'agit d'extraire les termes significatifs du document, c'est-à-dire de voir pour chaque terme rencontré, s'il ne fait pas partie d'une liste de mots vides (non utiles), exemple : les articles, les pronoms, ... etc, nommés aussi les " stop-words ". Si le terme est un stop-word il sera alors ignoré et ne sera pas pris pour indexer le document, Sinon il sera pris dans l'index.

- Calcul de la fréquence d'apparition du mot

Pour les mots significatifs, la fréquence d'occurrences doit être calculée. A chaque fois qu'un mot est rencontré dans le document, on incrémente sa fréquence (nombre d'apparition). Cette valeur obtenue est nommée *tf* (Term Frequency).

Le résultat de cette phase est le fichier " Index_Syn " : Index Syntaxique.
Il est constitué comme suit :
Pour chaque document, on trouve tous les mots qui appartiennent à ce document avec leurs fréquences d'occurrence.
N° document => {(terme$_1$, i fois),..., ($terme_n$, j fois)}, i,j : nombre d'occurrence du mot.
Comme exemple du résultat de cette étape, (voir la Figure 4.4)

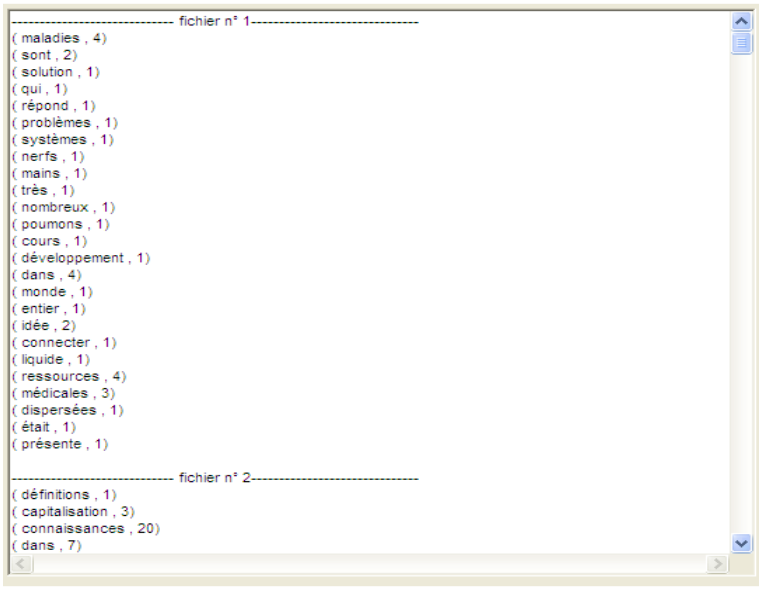

Fig. 4.4: Exemple d'index syntaxique

La phase de l'indexation syntaxique (élimination des stop-words et calcul du tf) se résume par l'algorithme suivant (Algorithm 1) :

```
    Require: docs : liste_Document
    Output: Index_Syn : index
1.1 i,j,k : entier ; /* i : Indice lignes ; j : Indice termes des ligne ;
    k : Indice fichiers ;                                              */;
1.2 mot_vide : booleen ; ligne : Ligne;
    /* Type Ligne (contient une liste de termes et une longueur)       */;
1.3 For k = 0 ; i < docs.nombre ; i++ do
        /* Faire ce traitement pour tous les fichiers                  */;
1.4     ouvrir_fichier_enCours();
1.5     For i = 0 ; i < fichier_enCours().longueur ; i++ do
            /* Faire ce traitement pour toutes les lignes              */;
1.6         ligne <- fichier_enCours().ligne(i);
            /* Récupérer la première ligne du fichier                 */;
1.7         For j = 0 ; i < ligne(i).nbr_termes ; i++ do
1.8             mot_vide <- Comparer(ligne[i].terme[j],Stop_List);
1.9             If mot_vide = False then
                /* Si le terme n'est pas un mot vide                  */;
1.10            If Index_Syn.Contient(ligne[i].terme[j]) = False then
                    /* Le terme n'existe pas dans l'Index_Syn         */;
1.11                Index_Syn.ajouter(ligne[i].terme[j],0);
                    /* tf=0 si terme n'existe pas dans index          */;
1.12            ELSE
1.13                Index_Syn.incrementer_tf(ligne[i].terme[j]);
                    /* Terme existe : incrémenter le tf               */;
1.14                END If;
1.15            END For;
1.16        END;
1.17    If fin_fichier() = true then
1.18        index_syn.New_ligne() ; /* Passer à la ligne pr doc suivant */
1.19    ;
1.20 END;
```

Algorithm 1: Indexation Syntaxique

L'algorithme précédent (Algorithm 1) se déroule ainsi :

- On fait le traitement suivant pour tous les fichiers :
- Ouvrir le fichier en cours, et pour toutes les lignes du fichier, faire le traitement suivant :
- Pour la ligne en cours, on compare tous ses termes avec la stop liste, pour déterminer si le terme est un mot vide.
- Si le terme n'est pas un mot vide et qu'il ne figure pas déjà dans l'index_Syn, on l'ajoute et son *tf* sera égal à 0. Sinon, si le terme figure dans l'index_Syn alors on incrémente son *tf* (sa fréquence d'apparition).
- Sinon si le terme est un mot vide, il sera alors ignoré.
- A la fin du fichier en cours, on fait un saut de ligne dans l'index_Syn, afin de stocker, s'il y en a, les termes du fichier suivant.

b- Indexation sémantique

Cette phase accepte en entrée les informations issues de l'indexation syntaxique. Elle est constituée des étapes suivantes (voir la Figure 4.5) :

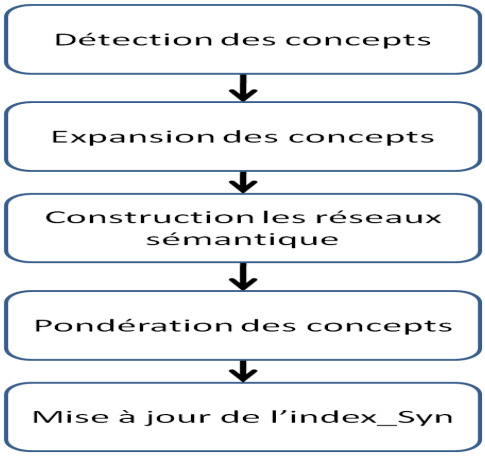

Fig. 4.5: Phases de l'indexation sémantique

Dans l'Index_Syn (issu de l'indexation syntaxique) et pour chaque document, on fait :

– **Détection des concepts**

Un terme est dit concept s'il appartient à au moins une entrée de l'ontologie.

La détection des concepts se fait en prenant les termes un à un et en les projetant sur l'ontologie pour détecter ce qui est concept de ce qui ne l'est pas.

Exemple : "Diabète" est un concept car il correspond à une entrée de l'ontologie qui est de type maladie alors que le terme "agir" par exemple ne sera pas pris comme concept.

A la différence de l'approche de Baziz et al. [26], le cas de détection des concepts formés par mots adjacents exemple : " infections pulmonaires ", n'est pas traité. On aura alors le terme " infections " qui sera pris comme un premier concept ainsi que " pulmonaire " qui sera pris comme un second concept.

Les concepts multi mots formés par l'approche de Baziz [26] sont rarement ambigus. Mais pour des raisons d'optimisation de la performance, d'amélioration de vitesse d'exécution et afin de diminuer la complexité de l'algorithme, nous avons opté pour cette démarche qui ne traite que les mots simples.

Dans cette étape, on fait la projection du document sur l'ontologie.

– **Expansion des concepts en utilisant l'ontologie**

Pour chaque concept détecté, un traitement spécifique est fait, il consiste à :

* Détecter les liens entre les différents concepts et de les relier ensemble en se basant sur l'ontologie.

* Etendre les concepts par leurs synonymes, dérivés et concepts de la même famille.

Une liste de synonymes et de sens sera attribuée à chaque concept, des liens et des relations entre concepts seront créés, ce qui forme une sorte de réseau pour chaque concept.

Dans cette étape on fait la projection de l'ontologie sur le document.

– **Construction du réseau sémantique**

Dans les deux étapes précédentes, nous avons utilisé notre ontologie pour représenter le contenu des documents sous forme de :

- Concepts.

- Relations entre concepts.

Nous avons eu comme résultat et pour chaque concept un nombre de termes qui lui sont associés. On appellera cette association le réseau sémantique du terme.

Nous définissons un réseau sémantique comme suit :

Définition

L'ensemble de termes R_k= { t_1,...,t_p } forme un réseau avec un terme t du document doc, c'est-à-dire :

- R_k est formé par l'union des éléments n_i de type terme qui sont en relation avec t, où relation ={ synonymie, dérivé, même_famille, ...}.

- ET $\forall\ t(n_i) \in R_k$, $t(n_i) \in doc$.

– **Pondération des concepts**

Il existe plusieurs approches pour pondérer les termes significatifs d'un document ou d'une requête. Nombre d'entre elles se basent sur les facteurs *tf* et *idf* qui permettent de considérer les pondérations locales et globales d'un terme.

Dans ce mémoire, on distingue la fréquence d'apparition d'un terme dans un document (term frequency, *tf*) qui a été déjà calculée dans l'indexation syntaxique et la fréquence d'apparition de ce même terme dans toute la collection considérée (inverse document frequency, *idf*).

La mesure *tf*idf* permet d'approximer la représentativité d'un terme dans un document, surtout dans les corpus de documents de tailles homogènes [27].

Dans notre approche, nous allons utiliser une variante de cette mesure qui le *Cf*idf*.

* Calcul du Cf

Le Cf est alors non pas la fréquence du terme mais celle du concept. Pour être calculé, on doit ajouter au *tf* du terme initial tous les *tf* des termes qui lui ont été associés dans le même document lors des phases précédentes, i.e les *tf* des termes du réseau sémantique.

Cf = *tf(t)* + $\sum$ *tf*(réseau). tel que t est le terme en cours et *tf*(réseau) sont tous les termes du réseau.

* Calcul du *idf*

Le *idf* est le nombre d'occurrences d'un terme dans tous les documents.

idf sera calculé comme suit : $idf(t) = \sum_{j=1}^{nbr_Doc} tf(t)$.

– **Mise à jour de l'index_Syn**

Pour faire la mise à jour de l'index_syn on ne prend en compte que les concepts ayant une entrée dans l'ontologie (c'est à dire les concepts ayant été détectés). Les termes qui appartiennent à leur réseau seront pris comme étant des concepts aussi avec la même pondération (qui est la somme) (enrichir l'index_Syn avec les termes du réseau créés ex : "synonymes").

On aura en résultat, un index sémantique (Index_Sem) qui est constitué comme suit :

N°document -> {{(concept_1$_k$, i), (concept_1$_{k+1}$,i)}, ... , {(concept_n_k,j), (concept_n_{k+1}, j)}}.

i,j : pondération du concept selon *Cf.idf* déjà calculé, k :1... m / m :entier.

Le résultat de cette étape est l'index sémantique (Index_Sem), il sera exploité par le service web S1 d'indexation.

La phase de l'indexation sémantique se résume par l'algorithme suivant (Algorithm 2) :

```
    Require: Index_Syn : index
    Output: Index_Sem : index sémantique
2.1 k : entier ; /* k : indice ;                                    */;
2.2 est_Concept : booleen ;
2.3 For k = 0 ; k < Index_Syn.nombre_Termes ; k++ do
        /* Faire ce traitement pour tous les termes de Index_Syn   */;
2.4     est_concept <– projection_sur_ontologie();
2.5     If est_concept = True then
            /* Le terme correspond à une entrée de l'ontologie     */;
2.6         expansion_avec_ontologie();
            /* Détecter les liens entre concepts et les étendre    */;
2.7         construction_réseau_sém();
            /* Le concept et les termes qui lui sont associés      */;
2.8         pondération_Concept();
            /* Le concept sera pondéré selon Cf.idf                */;
            /* Ces différentes étapes sont exécutées seulement si le
            terme est un concept                                    */;
2.9     Else /* Le terme sera alors ignoré                         */;
2.10    END If ;
2.11    MAJ_index_Syn();
        /* MAJ de l'index_Syn -> Seulement les concepts sont pris  */;
        /* On aura en sortie Index_Sem                              */
2.12 ;
2.13 END
```

Algorithm 2: Indexation Sémantique

L'algorithme précédent (Algorithm 2) se déroule ainsi :

- On fait le traitement suivant pour tous les termes de l'index_Syn (index syntaxique) :
- Projection du terme en cours sur l'ontologie.
- Si le terme est concept, faire le traitement suivant :
- Expansion du terme avec l'ontologie, ce que générera un réseau sémantique, on fait ensuite la pondération des concepts du réseau sémantique.
- Sinon si le terme n'est pas concept (ne correspond à aucune entrée de l'ontologie), il sera alors ignoré.
- La mise à jour de l'index_Syn est faite ensuite, en ignorant les termes non concepts, et en ajoutant les nouveau terme du réseau sémantique.

Création d'un index inversé

Cette étape n'est pas nécessaire pour l'indexation sémantique, mais elle est utile pour faire la création de l'index qui sera utilisé par le service web S2 d'indexation.
La création de l'index inversé consiste à : faire correspondre à tous les concepts, les documents qui les contiennent avec leurs pondérations, pour cela on doit regrouper tous les documents dans lesquels un concept apparait.

`Concept 1 −− > <document contenant le concept et sa pondération>`

`Concept 2 −− > <document contenant le concept et sa pondération >`

...

`Concept n −− > <document contenant le concept et sa pondération >`

Pour créer cette liste :

A partir de l'Index_Sem et pour tout concept de chaque document on fait un parcours de la liste qu'on veut créer et qui correspond aux concepts avec leurs documents, si le concept figure déjà on ajoute le document en cours aux autres documents contenant ce concept, sinon on ajoute le concept à la fin de la liste, ce processus est répété jusqu'à ce que tous les concepts de tous les documents soient traités.
Le résultat de cette étape (Index_Sem_Inv), l'Index Sémantique Inversé sera exploité par le service web S2 d'indexation.

L'algorithme suivant (Algorithm 3), permet de faire la création de l'index inversé :

```
Require: Index_Sem : index sémantique
Output: Index_Sem_Inv : index sémantique inversé
3.1 k, p : entier ; /* k : indice ;                                    */;
3.2 For k = 0 ; k < Index_Sem.nombreConcepts ; k++ do
        /* Faire ce traitement pour tous les concepts de tous les
        documents à partir de Index_Sem                               */;
3.3     p <– pondération ;
        /* Pondération du concept en cours dans le document           */;
3.4     If concept.enCours.existDans_Index_Sem_Inv() = True then
            /* si le concept existe déjà dans Index_Sem_Inv           */;
3.5         Index_Sem_Inv[concept.enCours].Ajout[num_Doc, p];
            /* ajouter le document en cours aux autres documents
            contenant ce concept dans Index_Sem_Inv avec pondération
            */;
3.6     Else Index_Sem_Inv[concept.enCours].Ajout[concept, num_Doc, p];
        /* ajouter à l'index : concept, document et pondération       */;
3.7     END If;
3.8     END
```

Algorithm 3: Création de l'index inversé

L'algorithme précédent (Algorithm 3) se déroule ainsi :

- On fait le traitement suivant pour tous les documents, à partir de l'Index_Sem (Index sémantique) :
- Faire ce traitement pour tous les concepts :
- Si un concept apparait déjà dans l'index inversé, alors ajouter le document en cours avec la pondération du concept à l'index inversé (Index_Sem_Inv).
- Sinon si un concept n'apparait pas déjà dans l'index inversé, alors ajouter le concept, le document en cours et la pondération du concept à l'index inversé (Index_Sem_Inv).

4.1.2 Indexation manuelle

Comme nous avons dit précédemment, ce type d'indexation est fait manuellement, c'est-à-dire que les documents seront annotés manuellement par des médecins (des utilisateurs) sur les ressources non textuelles, ex : une vidéo, une image (Figure 4.6).

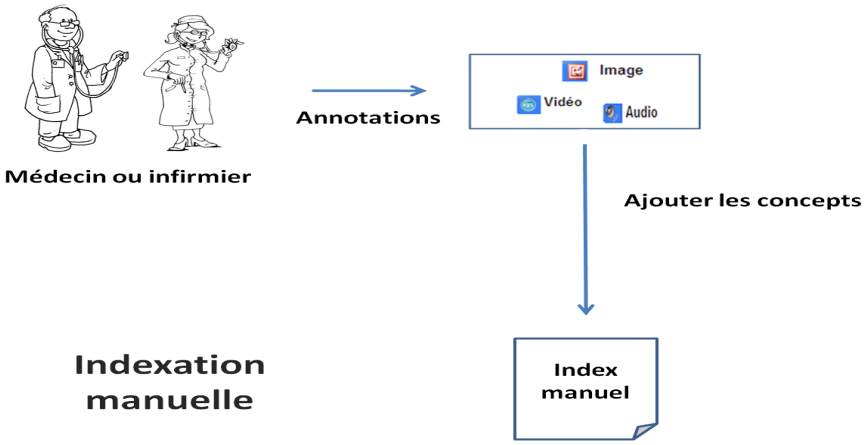

Fig. 4.6: Indexation manuelle

Création de l'index Sémantique manuel

Après l'annotation manuelle d'un fichier non textuel, faite par un spécialiste (médecin par exemple) en se basant sur les concepts d'une ontologie, ces derniers (les concepts) seront stockés dans un index et serviront pour indexer le fichier. On considère que pour l'indexation manuelle, un fichier est pertinent pour un concept, s'il est annoté par ce concept et qu'on n'a pas la notion de tf et $Cf.IDF$. Le fichier sera alors retourné en résultat s'il contient le concept recherché.

On aura alors pour un fichier f, un ensemble de couple (concept, instance) :

`N°fichier -> {{concept_1, instance }, ..., { concept_`n`, instance}}.`

Le résultat de cette étape est un index manuel (Index_Sem_Man).

Remarque :

Plusieurs travaux de recherche ont été réalisés dans le cadre de l'indexation manuelle basée sur les ontologies, on peut citer :

- Un système d'indexation sémantique des images cérébrales qui a été réalisé par Gaouar et al. [28].
- La conception et l'utilisation d'ontologies pour l'indexation de documents audio-visuels par A. Isaac [6].

Ces travaux peuvent être exploités, pour une éventuelle amélioration de notre travail.

Cr§ation de l'index S§mantique manuel invers§

Pour chaque instance de concept, on recherche les documents qui ont été annotés par cette même instance, et on les regroupe, ce qui permet d'avoir :

```
Instance 1 -> <documents contenant l'instance>
...
Instance n -> <documents contenant l'instance>
```

Le résultat de cette étape est un index manuel inversé (Index_Sem_Man_inv).

4.2 Recherche dans l'index

Quatre index seront utilisés par les services web d'indexation :
Pour l'indexation automatique, le service web S1 utilisera (Index_Sem) et le service web S2 utilisera (Index_Sem_Inv).
Pour l'indexation manuelle, le service web S1 utilisera (Index_Sem_Man) et le service web S2 utilisera (Index_Sem_Man_inv).

Pour les deux services web, la recherche est effectuée dans les index manuels et dans les index automatiques. La recherche se fait en évaluant la requête avec le respect d'un seuil et la prise en considération des pondérations dans le cas de l'indexation automatique.

1. Index_Sem (Service S1) : permet de faire entrer un code d'un document (un fichier), et la liste des concepts que contient ce document sera retournée et ordonnée par la pondération pour l'indexation automatique après avoir filtrer les concepts ayant une pondération inférieure à un certain seuil. Les fichiers indexés manuellement sont retournés automatiquement (Index_Sem_Man).

2. Index_Sem_inv (Service S2) : permet de faire entrer un concept, et la liste de documents qui concerne le concept sera retournée, ils seront ordonnés par ordre décroissant du *tf* (fréquence d'apparition du concept dans le document).
   On considère pour l'index manuel inversé (Index_Sem_Man_inv) qu'un document est automatiquement retourné et qu'il est pertinent pour le concept, car il a été annoté manuellement par un expert.

4.3 Conclusion

Cette dernière partie nous a permis de détailler et de mettre le point sur les fonctionnalités de l'approche proposée. Nous avons commencé par la présentation de l'architecture du système global de médiation, afin de situer notre partie. Nous avons donné ensuite un exemple d'ontologie de médiation, à partir duquel les services web d'indexation que nous avons proposé ont été décrits, sous formes de vues RDF pour permettre leur intégration avec les services web d'interrogation dans le système global. Nous avons également décrit les différentes étapes de la construction des index exploitables par nos services web d'indexation à l'aide de diagrammes. Des algorithmes sont présentés aussi pour décrire les différentes étapes de l'approche d'indexation proposée. Le chapitre suivant permettra de concrétiser notre conception par la présentation des étapes d'implémentation d'un prototype.

Troisième partie

Prototype

5

Conception d'un prototype

5.1 Introduction

Afin de concrétiser et de valider notre approche d'indexation et d'utilisation des deux services web proposés, nous avons conçu un prototype reflétant le fonctionnement de notre proposition, nous avons réalisé une première version de ce prototype dont les résultats font l'objet du présent chapitre. Nous commencerons d'abord par décrire l'environnement dans lequel nous avons réalisé le prototype puis nous discuterons les résultats que nous avons obtenu.

5.2 Environnement de développement

Nous avons développé notre application sur une machine Intel Core2duo, avec une vitesse de 1.6 GHZ, dotée d'une capacité mémoire de 2GB de RAM sous Windows XP. L'application est développée en utilisant le langage de programmation Java.

5.3 Pourquoi java ?

Java est un langage de programmation orienté objet et un environnement d'exécution récent, développé par Sun Microsystems en 1991.
Java possède de nombreuses caractéristiques qui font de lui un langage de choix, il permet de développer des applications client-serveur. Coté client, les applets sont à l'origine de la notoriété du langage. C'est surtout coté serveur que java s'est imposé dans le milieu de l'entreprise grâce aux servlets, le pendant serveur des applets, et plus récemment les JSP (Java Server Pages) qui peuvent se substituer à PHP, ASP et ASP.NET [29].

5.4 Description du fonctionnement de notre prototype

Dans cette partie, nous allons présenter les interfaces de notre application à travers un exemple qui a été réalisé.

5.4.1 Accès à l'application

La Figure 5.1 est la première interface de notre prototype qui apparaît à l'utilisateur, elle représente l'interface du lancement du prototype.

Fig. 5.1: Première interface de notre prototype

Cliquer sur " Continuer " permet le lancement du prototype, une autre interface apparait, elle représente la fenêtre principale de notre prototype (voir Figure 5.2).

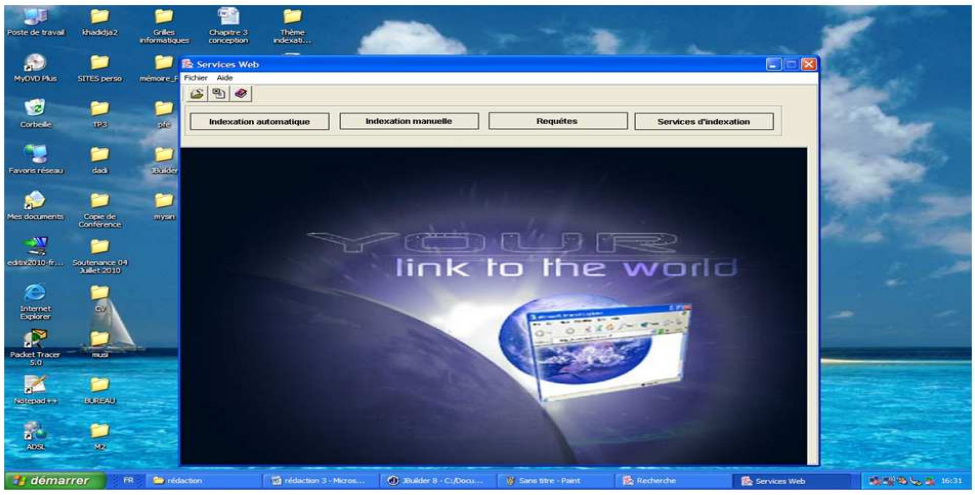

Fig. 5.2: Fenêtre principale de notre prototype

La barre du menu contient deux menus Fichier et Aide. Chaque menu est composé d'un ensemble d'Items. Des raccourcis vers les menus les plus importants, existent sous forme de boutons au menu principal (voir Figure 5.3).

Fig. 5.3: Boutons du Menu principal

A partir du menu principal (Figure 5.3), l'utilisateur a le choix entre quatre possibilités :

1- Lancer une indexation automatique

Ça consiste à faire une indexation sémantique automatiquement, à un corpus de fichiers (un répertoire).

2- Lancer une indexation manuelle

L'indexation manuelle, permet d'annoter manuellement les fichiers non textuels, tels que les images.

3- Écrire une requête

Cela permet de rendre une réponse à une requête plus ou moins complexe, en faisant une combinaison des réponses partielles des termes composant la requête.

4- Utiliser les services web d'indexation

L'utilisation des services web permet d'exploiter les index créés par les phases d'indexation précédentes.

5.4.2 Utilisation du prototype

Le prototype a pour but d'étudier la validité de l'approche proposée. Nous allons présenté dans cette partie quelques captures d'écran des utilisations de notre prototype.

1. **L'indexation automatique**

Si l'utilisateur clique sur le bouton " Indexation automatique " du menu principal (Figure 5.3), une interface apparait, offrant la possibilité de lancer l'indexation automatique d'un ensemble de fichiers (voir Figure 5.4).

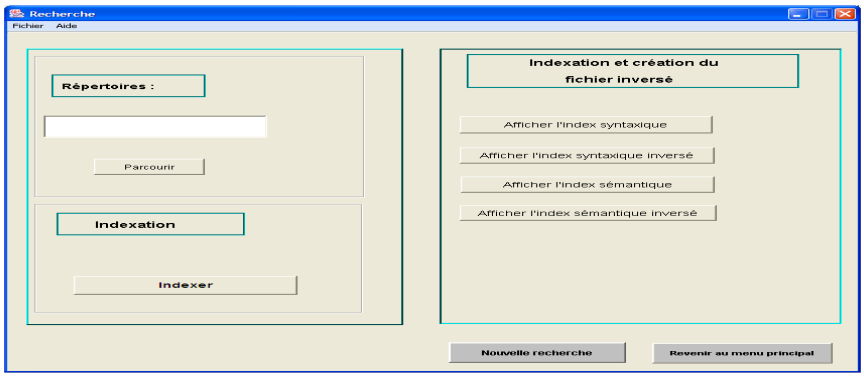

Fig. 5.4: Interface d'indexation automatique

L'utilisation de cette interface est comme suit (Figure 5.5) :

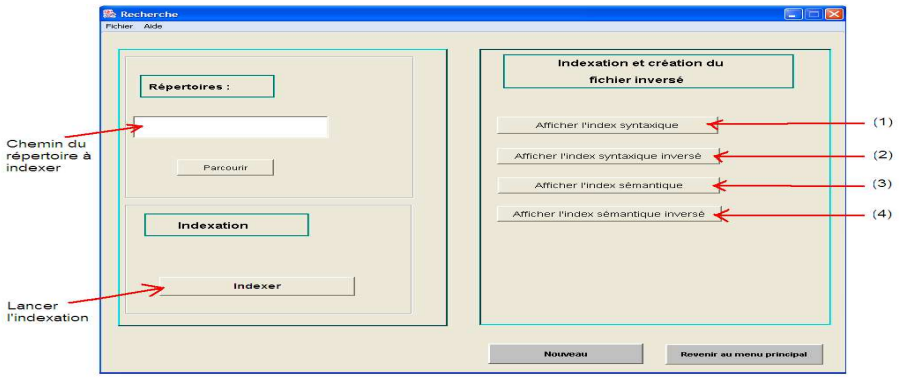

Fig. 5.5: Utilisation de l'interface d'indexation automatique

Dans la partie gauche de l'interface (Figure 5.5), l'utilisateur peut sélectionner le chemin du répertoire à indexer, en cliquant sur le bouton " Parcourir ", il peut ensuite lancer l'indexation en cliquant sur " Indexer ".

Dans la partie droite de l'interface (Figure 5.5), l'utilisateur peut visualiser les différents index créés lors de la phase d'indexation. Il peut afficher :

(a) L'index syntaxique : créé lors de la phase de l'indexation syntaxique (Index_Syn).

(b) L'index syntaxique inversé : c'est le résultat inversé de l'index créé par la phase d'indexation syntaxique.

(c) L'index sémantique : créé lors de la phase d'indexation sémantique à partir de l'index syntaxique (Index_Sem).

(d) L'index sémantique inversé : créé de la phase de construction de l'index sémantique inversé (Index_Sem_Inv).

2. **L'indexation manuelle**

Si l'utilisateur clique sur le bouton " Indexation manuelle " du menu principal (Figure 5.3), une interface apparait, offrant la possibilité de faire manuellement l'indexation des fichiers non textuels (voir Figure 5.6).

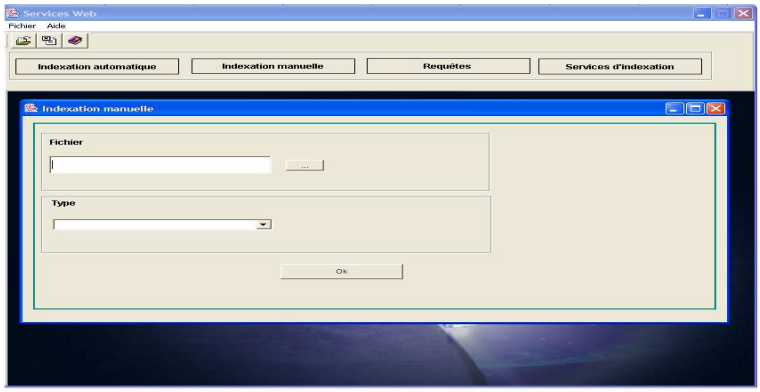

Fig. 5.6: Interface d'indexation manuelle

L'utilisation de l'interface est comme suit (voir Figure 5.7) :

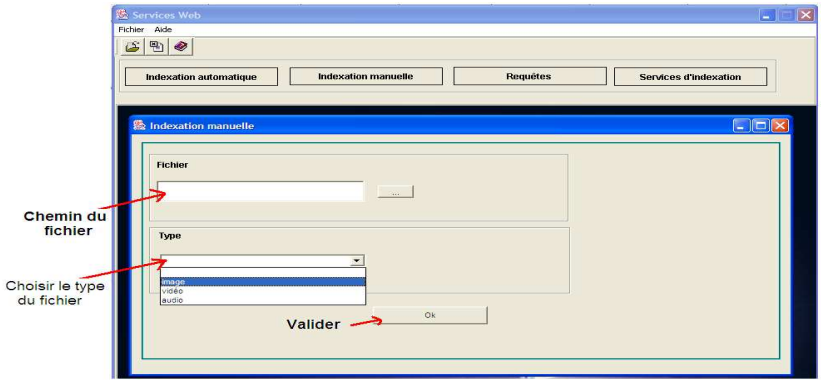

Fig. 5.7: Utilisation de l'interface d'indexation manuelle

L'utilisateur doit indiquer le chemin du fichier qu'il veut indexer manuellement, et il choisit son type : (image, vidéo, audio), chaque type a son traitement spécifique.

3. **Les requêtes**

Si l'utilisateur clique sur le bouton " Requêtes " du menu principal (Figure 5.3), une interface apparait, elle offre la possibilité de faire la saisie d'une requête (voir Figure 5.8).

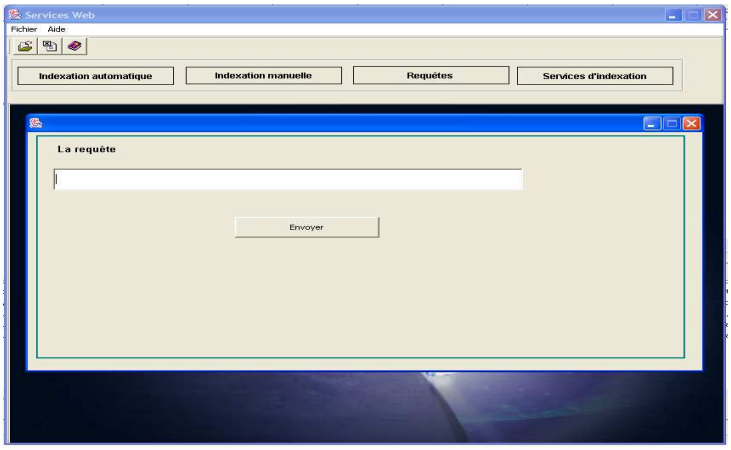

Fig. 5.8: Interface des requêtes

Pour cela, une zone est réservée pour faire la saisie de la requête.

Cliquer sur " Envoyer " permet de valider la saisie de la requête (Figure 5.9).

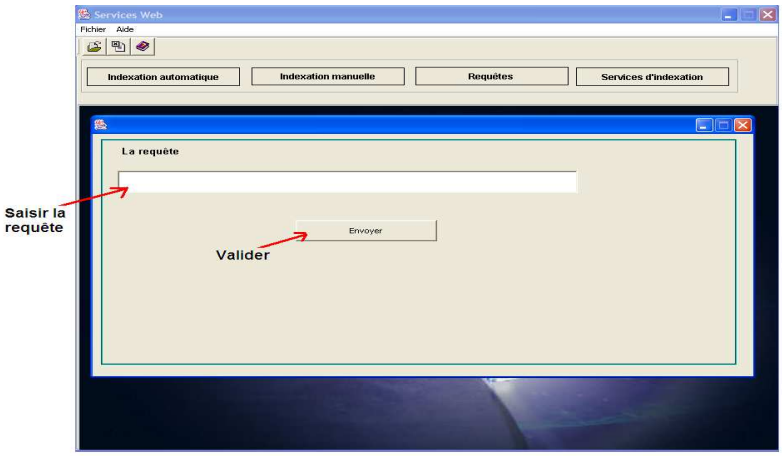

Fig. 5.9: Utilisation de l'interface des requêtes

4. **Les services web**

Si l'utilisateur clique sur le bouton " Services d'indexation " du menu principal (Figure 5.3), une interface apparait, elle offre la possibilité de tester les deux services web d'indexation que nous avons proposé (un services pour avoir comme résultat les fichiers à partir des concepts, et un service pour avoir les concepts à partir d'un fichier) (voir Figure 5.10).

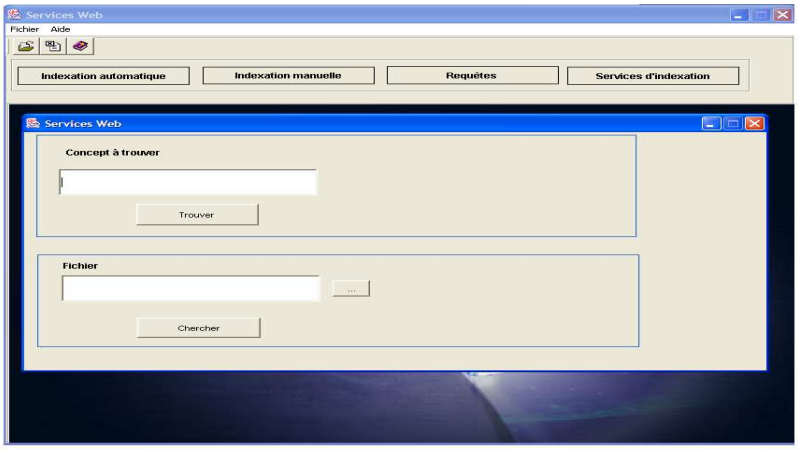

Fig. 5.10: Services Web d'indexation

L'explication de l'utilisation des deux services web est représentée par la Figure 5.11 :

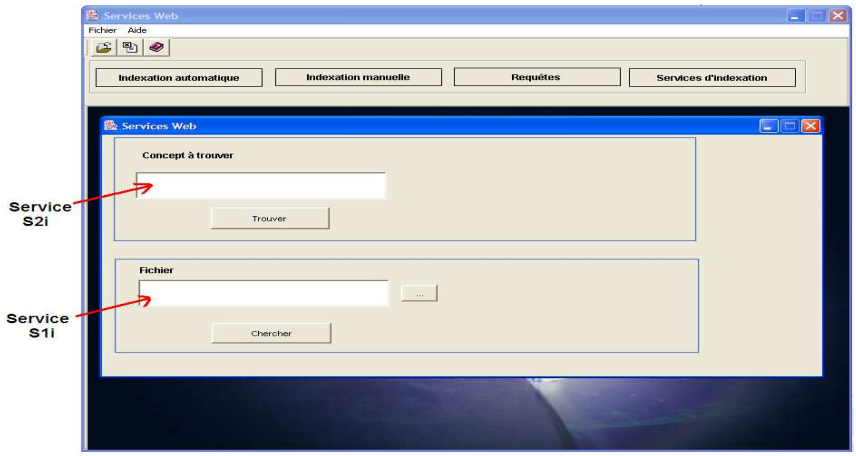

Fig. 5.11: Utilisation de l'interface des services Web d'indexation

Dans la Figure 5.11 :

1. On peut remarque que l'utilisateur peut saisir le concept qu'il souhaite rechercher dans la zone réservée au 'Concept'. En cliquent sur " Trouver ", une recherche est lancée afin de chercher le concept et retourner tous les fichiers contenant ce concept. La recherche est faite dans l'index sémantique inversé (Index_Sem_Inv) et dans l'index sémantique manuel inversé (Index_Sem_Man_Inv) , car ce sont les index qui contiennent les concepts avec leurs fichiers correspondants.
   Ce traitement représente le fonctionnement du service web S2i d'indexation (comme il a été définit dans la partie conception).
2. L'utilisateur peut aussi indiquer le chemin d'un fichier dans la zone 'Fichier'. En cliquant sur " Chercher ", une recherche est lancée afin de trouver tous les concept du fichier sélectionné. La recherche est faite dans l'index sémantique (Index_Sem) et dans l'index sémantique Manuel (Index_Sem_Man), car ce sont ces index qui contiennent tous les fichiers avec les concepts qui leurs indexent.
   Ce traitement représente le fonctionnement du service web S1i d'indexation (comme il a été défini dans la partie conception).

5.5 Conclusion

Dans ce chapitre, nous avons présenté notre prototype, i.e une application qui démontre la faisabilité de l'approche d'indexation et l'utilisation des deux services web proposés. Nous avons commencé par décrire le langage de programmation utilisé pour son développement ainsi que quelques interfaces du prototype. Nous avons également montré un exemple d'utilisation du prototype avec ses différents étapes. Nous notons que dans un travail ultérieur, cette application pourra être intégrée dans un JSP (Java Server Pages) ou applet pour qu'elle puisse être destinée au web, et donc traiter les données distribuées.