Élaboration 2 du :

META-MOTEUR DE RECHERCHE SUR LE WEB ORIENTE COMMUNAUTE

Équipe :

M. Jérémy FRÉCHARD
Mlle Cécile GIRARD
Mlle Aysel GUNES
M. Pierre RAMOS
Étudiants en Master Informatique première année

Client :
M. Rushed KANAWATI

Encadrant :
Mme Sophie TOULOUSE

(26/04/2006)
Version 1.0

Sommaire

I/ IHM

1) Caractéristiques de l' IHM du méta-moteur

L' IHM du méta-moteur est l'interface privilégiée de l'utilisateur. Elle doit donc à la fois répondre totalement à ses exigences mais aussi simplifier au maximum la recherche.

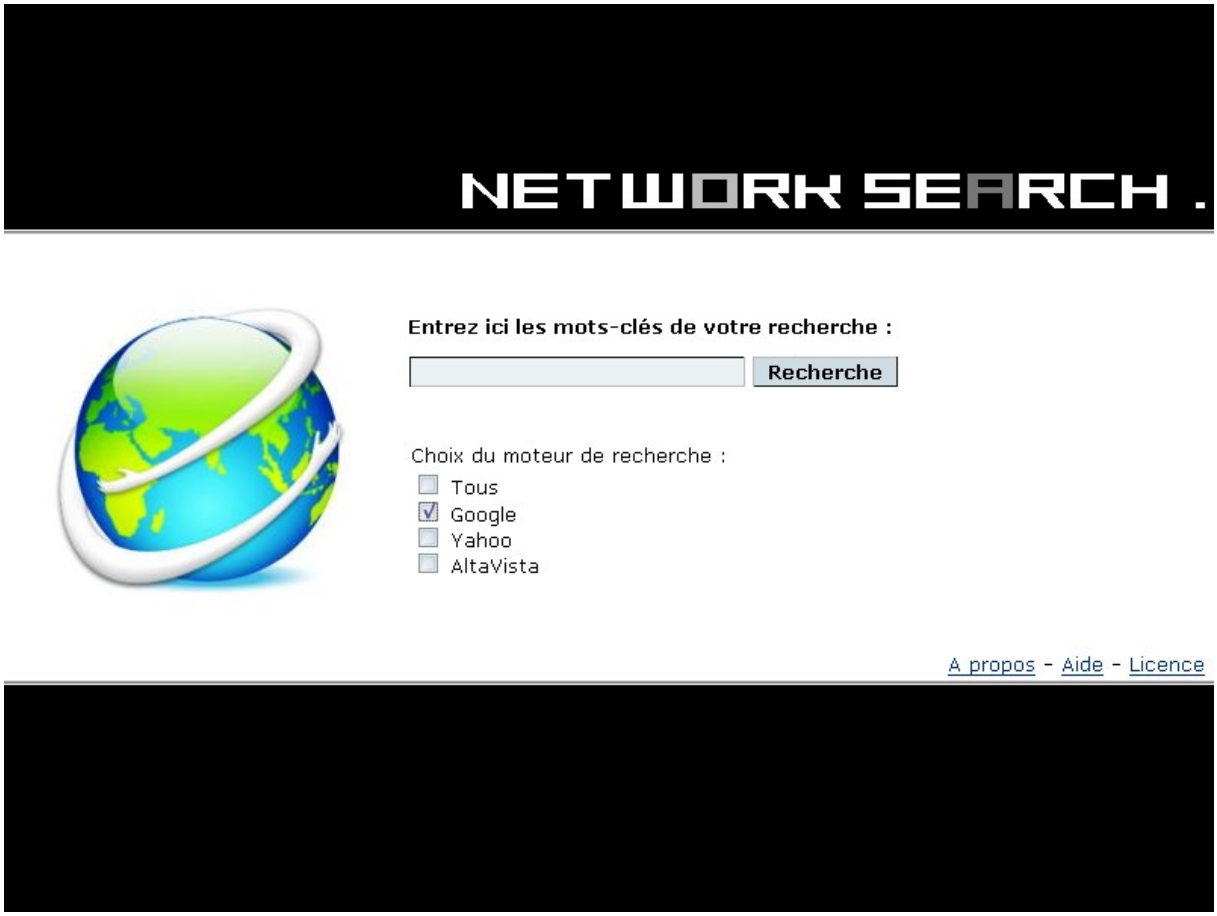
Nous avons donc opté pour une interface épurée, claire et ergonomique mettant en avant les principales fonctions du méta-moteur, à savoir : la recherche et les options de recherche.

Caractéristiques de l'interface :

- Valide XHTML 1.0 Strict
- Valide CSS 2.0
- Respect des standards d'accessibilité

2) Design de l'interface

Cette page constituera l'index de recherche, ce sera la page principale du méta-moteur.

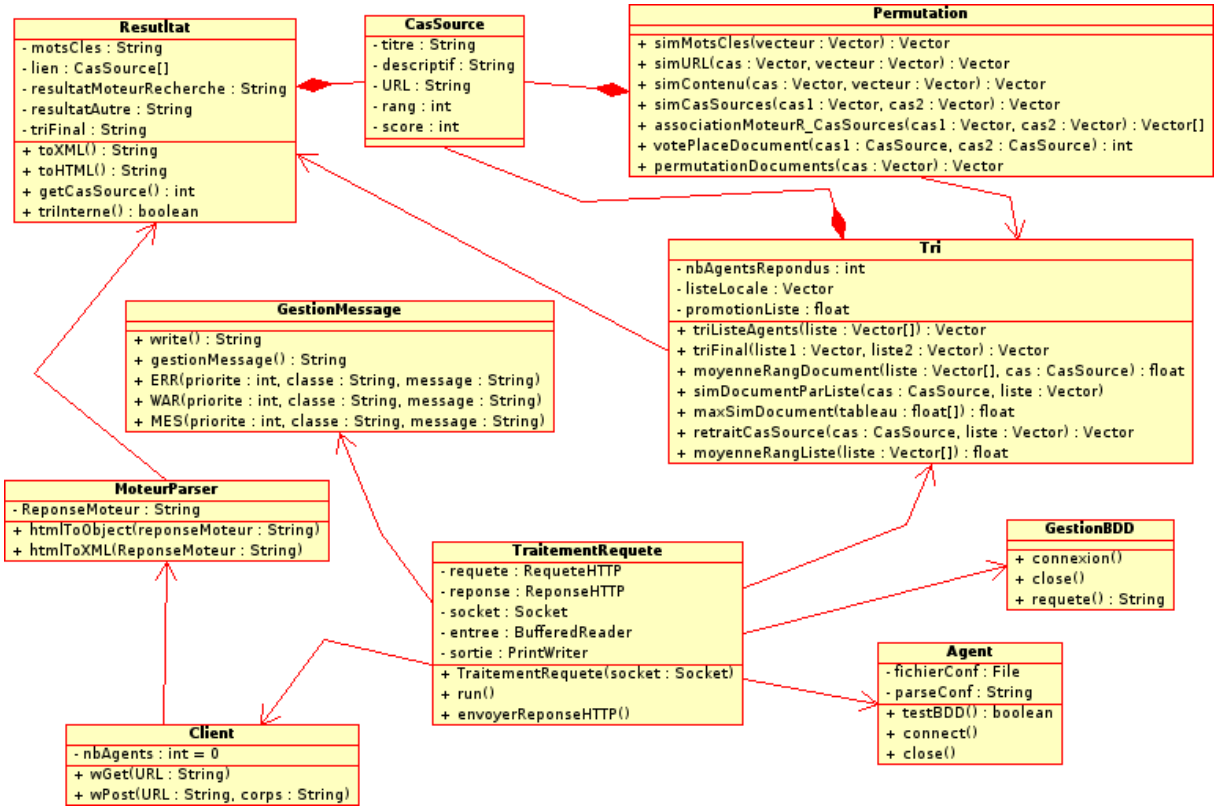


Voici la page d'affichage de résultats d'une recherche effectuée par un utilisateur. Cette page sera de largeur fluide permettant ainsi un affichage optimisé sur les différentes résolutions d'écran. Un maximum de 20 liens (et descriptifs) seront proposés, en accord avec le cahier des charges. Il n'est donc pas nécessaire de proposer un lien du type « suite des résultats ».

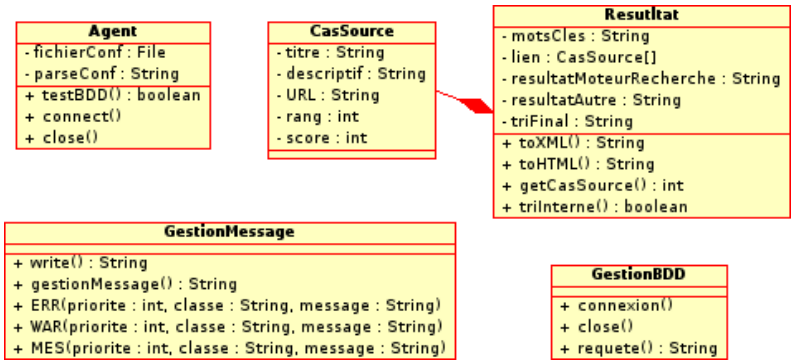
Cette mise en page quant à elle constituera le design des pages connexes (A propos, Aide et Licence).

II/ Architecture logicielle

1) Diagramme général simplifié du méta-moteur



2) Diagramme de classe des Objets principaux



La classe **Agent** : c'est la classe principale du méta-moteur celle qui sera exécuter au lancement de l'application.

1. Parsing du fichier de configuration par la classe associée
2. Connexion à la base de données
3. Lancement du serveur
4. Boucle du programme
5. Arrêt du serveur

La classe **GestionMessage** :

Elle gère les messages interne de l'agent et permet soit d'écrire sur la sortie standard le message, soit d'enregistrer le message dans un fichier log ou bien les 2 en même temps. Les messages sont de trois types différents : - [ERR] les erreurs du système
- [WAR] les erreurs non-bloquantes
- [MES] les messages de traitements

Les messages sont écrit de la manière standardisé suivante : [ERR] date - heure - classeEmettrice – Message

La méthode ***Write*** permet d'écrire le message soit dans le fichier log, soit en sortie standard
La méthode ***gestionMessage*** permet de déclenchée la méthode adaptée suivant les configuration de l'agent.

Dans la classe **Résultats** :

Elle permet de réunir la liste de résultats que l'utilisateur obtiendra

La méthode ***ToXML*** permet de transformer l'objet résultat en XML.
La méthode ***ToHTML*** permet de transformer l'objet résultat en HTML.
La méthode ***GetCasSource*** permet d'obtenir les CasSources d'un résultat.
La méthode ***TriInterne*** permet de tester si le tri à effectuer est oui ou non un tri interne, c'est-à-dire que les autres agents n'ont pas de tri à réaliser sur la requête.

Dans la classe **GestionBDD** :

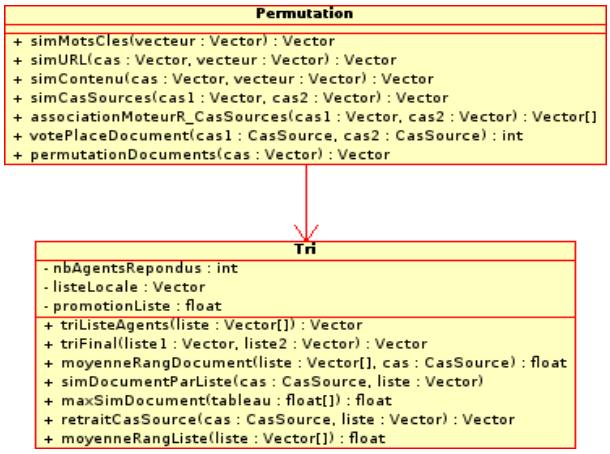
Elle permet de gérer la base de données (connexion, fermeture et requête sur la base)

La méthode ***Connexion*** permet de se connecter à la base de données
La méthode ***Close*** permet de fermer la base de données
La méthode ***Requête*** permet d'exécuer une requête sur la base de données

Dans la classe **CasSource** :

Elle permet de décrire ce qu'un lien possède tels qu'un titre, un descriptif, une URL, un score et un rang.

3) Diagramme de classe pour le module de tri



Deux classes seront suffisante pour le module de TRI :
La classe ***Permutation*** : Elle permettra d'effectuer une permutation des résultats moteur de recherche.
La classe ***Tri*** : Elle permettra de réaliser le tri final.

Méthodes de la classe Permutation

simMotsCles

En entrée : Les mots clés entrés par l'utilisateur.

En sortie : Un vecteur de Cas Sources.

Cette méthode permettra de récupérer dans la base de données de l'agent en question tous les cas sources dont les mots clés sont similaires à ceux entrés par l'utilisateur.

simURL

En entrée : - Une URL correspondant à l'une des URL retournées par le moteur de recherche.
- Le vecteur de Cas Sources retenu par la méthode ***simMotsCles***.

En sortie : Un vecteur de Cas Sources.

Cette méthode permettra de récupérer, parmi les cas sources retenus lors de l'appel à la méthode ***simMotsCles,*** les cas sources dont les URL sont similaires à celles passée en paramètre (ce qui correspond à l'une des URL retournée par le moteur de recherche).

simContenu

En entrée : - Le descriptif de l'une des réponses du moteur de recherche.
- Le vecteur de Cas Sources retenu par la méthode ***simMotsCles***.

En sortie : Un vecteur de Cas Sources.

Cette méthode permettra de récupérer, parmi les cas sources retenus lors de l'appel à la méthode ***simMotsCles,*** les cas sources dont les contenus du descriptif sont similaires à ceux passés en paramètre.

simCasSources

En entrée : - Le vecteur de Cas Sources retourné par le moteur de recherche.
- Le vecteur de Cas Sources retenu par la méthode ***simMotsCles***.

En sortie : Un vecteur de Cas Sources.

Cette méthode permettra de récupérer tous les cas sources similaires retenus par la méthode ***simMotsCles*** par rapport aux cas sources du moteur de recherche.

associationMoteurR_CasSources

En entrée : - Le vecteur de Cas Sources retourné par le moteur de recherche.
- Le vecteur de Cas Sources retenu par la méthode ***simCasSources***.

En sortie : - vector<CasSources>[1..2]

Cette méthode permettra d'associer à chaque cas sources retourné par le moteur de recherche un cas source retenu pas la méthode ***simCasSources*** (s'il n'en existe pas, alors le champ sera initialisé à NULL).

votePlaceDocument

En entrée : - Un cas source appartenant à ceux retournés par le moteur de recherche.
- Le cas source appartenant à ceux retenus par la méthode ***simCasSources*** et associé au cas source donné en premier argument.

En sortie : Un entier.

Cette méthode permettra de décider de la place d'un cas source, d'un document dans la liste à retourner à l'utilisateur.

permutationDocuments

En entrée : Le vecteur de Cas Sources retourné par le moteur de recherche.

En sortie : Ce même vecteur de Cas Sources avec la place des documents ayant changée selon la réponse de la méthode ***votePlaceDocument***.

Cette méthode permettra de reclasser la place des documents (retournés par le moteur de recherche) selon les cas sources enregistrés dans la base de données de l'agent en question.

Instances de la classe Tri

nbAgentsRepondus

Un entier *(une variable)* correspondant au nombre d'agents ayant répondu, c'est-à-dire ayant renvoyé une liste de résultats.

listeLocale

Un vecteur de Cas Sources *(une variable statique)* correspondant au vecteur de Cas Sources de l'agent local (celui qui a fait la demande de recherche auprès des autres agents).

PROMOTION_LOCALE

Un réel *(une constante)* correspondant à une constante qui permettra de promouvoir la liste de résultats locale.

Méthodes de la classe Tri

triListeAgents

En entrée : Un tableau de vecteurs de Cas Sources (tous les vecteurs de cas sources retournés par chaque agent contacté et ayant répondu).
En sortie : Un vecteur de Cas Sources.

Cette méthode permettra de regrouper tous les vecteurs de cas sources que chaque agent contacté a renvoyé sous la forme d'un vecteur de Cas Sources.

triFinal

En entrée : - le vecteur de Cas Sources regroupé grâce à la méthode ***triListeAgents.***
- Le vecteur de Cas Sources local (à savoir celui de l'agent local).
En sortie : Un vecteur de Cas Sources (la liste de résultats finale).

Cette méthode permettra de trier, de reclasser la liste de résultats locale selon la liste de résultats regroupée des autres agents.

moyenneRangDocument

En entrée : - Un Cas Source de l'une des listes de résultats retournées par les autres agents (il faut savoir que l'on gardera cette même liste ; on fera une boucle sur les cas souces de cette liste afin de les parcourir tous).
- Un tableau de vecteurs de Cas Sources aux autres listes de résultats (sauf celle choisie en premier argument).
En sortie : Un réel.

Cette méthode permettra de faire une moyenne du rang d'un cas source : sommer le rang de ce cas source dans chaque liste retournée par chaque autre agent et diviser le tout par le nombre d'agents ayant répondus.

simDocumentParListe

En entrée : - Un Cas Source de l'une des listes de résultats retournées par les autres agents (il faut savoir que l'on gardera cette même liste ; on fera une boucle sur les cas souces de cette liste afin de les parcourir tous).
- Un vecteur de Cas Sources.
En sortie : Un tableau de réels.

Cette méthode permettra de calculer la similarité entre un cas source et chaque cas source d'un vecteur de Cas Sources.

maxSimDocument

En entrée : Un tableau de réels.
En sortie : Un réel.

Cette méthode permettra de retourner le maximum dans un tableau de réels.

retraitCasSource

En entrée : - Le Cas Source à retirer.
- Le vecteur de Cas Sources sur lequel doit s'effectuer le retrait.

En sortie : Un vecteur de Cas Sources qui ne contiendra plus le cas source que l'on désirait retirer.

Cette méthode permettra de retirer d'un vecteur de Cas Sources de retirer un cas source donné en argument.

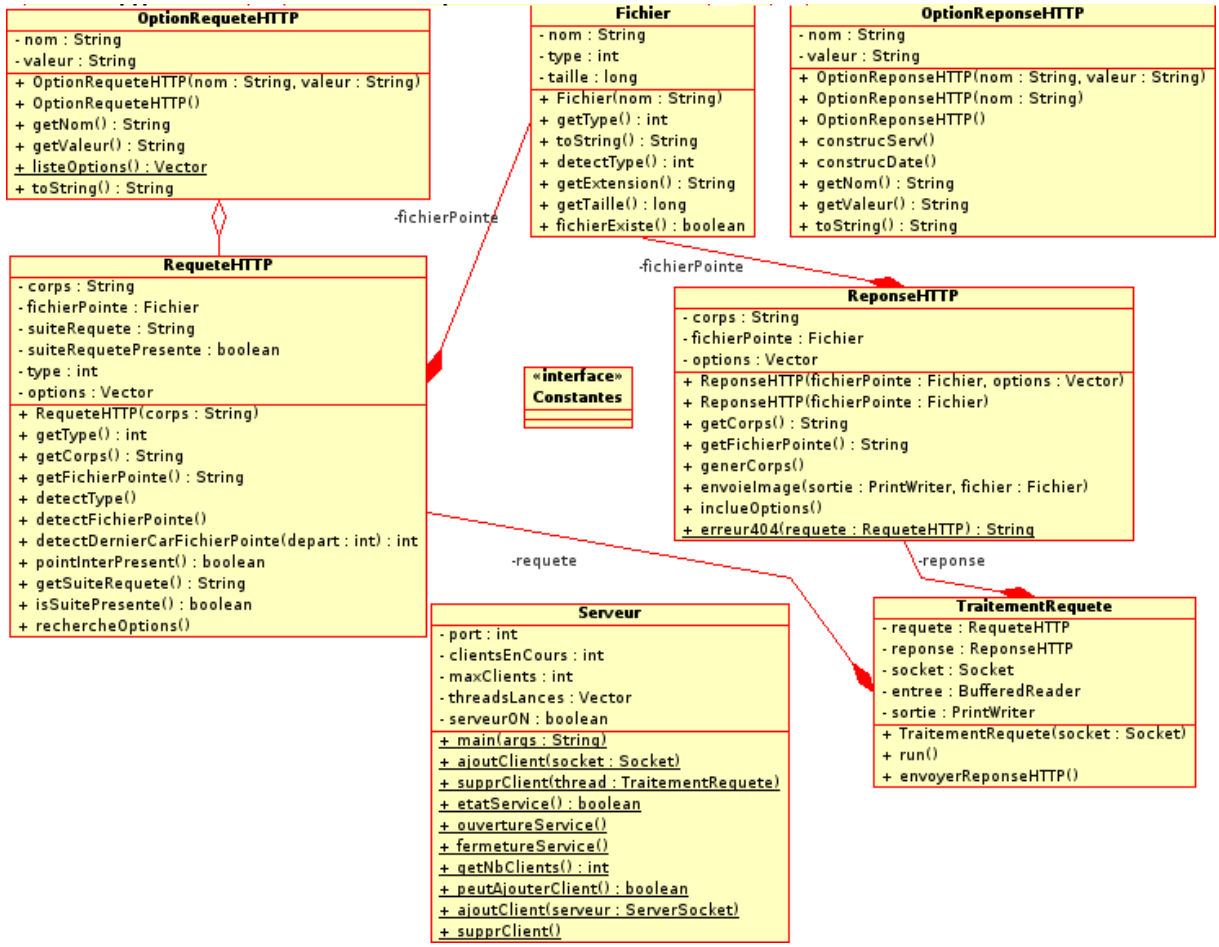
moyeneRangListe

En entrée : Un tableau de vecteur de Cas Sources.
En sortie : Un tableau de réels.

Cette méthode permettra d'obtenir la moyenne des rangs de chaque cas source d'une liste. Nous aurons donc un tableau de réels correspondant au rang de chaque cas source.

4) Diagramme de classe pour le module de connexions

a) Partie Serveur



7 classes sont nécessaires pour le module de connexions partie client ainsi qu'une interface :
l'interface Constantes : cette interface contient uniquement les constantes utilisées par les autres méthodes du serveur au format entier :

GET
IMAGE_BMP
IMAGE_GIF
IMAGE_JPG

IMAGE_PNG
INCONNU
POST
TAILLE_BLOC
TEXTE

Les Classes :

La classe **Fichier** : Elle permet de représenter un fichier et d'en manipuler certains aspects
La classe **OptionRequeteHTTP** : Elle permet de récupérer les options qu'un client peut envoyer dans sa requête
la classe **OptionReponseHTTP** : Elle permet de générer les options que le serveur peut envoyer au client lors de la réponse a la requête
la classe **ReponseHTTP** : Elle permet de générer une réponse adaptée a la requête envoyée par le client
la classe **RequeteHTTP** : Elle permet de parser la requête envoyée par le client
la classe **TraitementRequete** : Elle permet de lancer un thread permettant de traiter une requête envoyée par un client
la classe **Serveur** : Elle permet d'attendre les connections des clients et d'en gérer le nombre

méthodes de la classe **Fichier**:
getType
En sortie : le type du fichier (sous forme de constante)
toString
En sortie : le nom du fichier

méthodes de la classe **OptionRequeteHTTP**:
getNom
En sortie : le nom de l'option
getValeur
En sortie : la valeur de l'option
listeOptions
En sortie : un vecteur contenant les options reconnues par le serveur
toString
En sortie : une chaîne de caractères représentant l'option

méthodes de la classe **OptionReponseHTTP**:
construcServ
construit automatiquement une option de type Server
construcDate
construit automatiquement une option de type Date
getNom
En sortie : le nom de l'option
getValeur
En sortie : la valeur de l'option
toString
En sortie : une chaine de caractères représentant l'option

méthodes de la classe **ReponseHTTP**:
getCorps
En sortie : le corps de la réponse
getFichierPointe
En sortie : une chaîne de caractères contenant le nom du fichier pointé par la réponse
envoieImage
E*n entrée :* un Fichier
un PrintWriter

cette méthode permet d'envoyer directement un fichier sur le flux d'envoi au client (ceux-ci sont passés en argument)

erreur404

En entrée : une RequeteHTTP

En sortie : une chaîne de caractères a envoyer au client (contenant le corps de la réponse qui lui sera transmise)

cette méthode permet de générer automatiquement une page d'erreur 404 lorsqu'un client demande un fichier non existant.

méthodes de la classe **RequeteHTTP**:

getCorps

En sortie : le corps de la requête

getFichierPointe

En sortie : une chaîne de caractères contenant le nom du fichier pointé par la requête

getType

En sortie : le type de la requête (sous forme d'entier dans Constantes)

getSuiteRequete

En sortie : une chaîne de caractères contenant la suite de la requête (qui suit me ? pour une requête Get et le corps de la requête pour une requête Post)

isSuitePresente

En sortie : true si il y a une suite a la requête, false sinon

méthodes de la classe **TraitementRequete**:

run

cette méthode est la surcharge de la méthode run de Thread, cette méthode reçoit puis traite une requête envoyée par un client

méthodes de la classe **Serveur**:

ajoutClient

en entrée : le socket sur lequel cette méthode doit attendre les connexions de nouveaux clients

supprClient

méthode permettant de supprimer un client (lorsque celui-ci a terminé son traitement

etatService

En sortie : true si le client doit rester en fonction, false sinon

ouvertureService

initialise les variables nécessaires au démarrage du serveur

fermertureService

force la fermeture du serveur

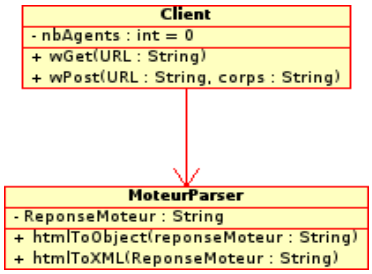
getNbClients

En sortie : le nombre de client actuellement en cours de traitement

startServeur

méthode startServeur permettant de lancer le serveur

b) Partie Client



La classe **Client** :

Elle permet de contacter d'autres agents ou des moteurs de recherches par des requêtes HTTP.
La méthode ***wGet*** implémente la commande GET côté client du protocole HTTP
La méthode ***wPost*** implémente la commande POST côté client du protocole HTTP

La classe **MoteurParser** :

Cette classe formate la réponse d'un moteur de recherches et la transforme en un type utilisable par d'autres classes (notamment celles relative aux tris).
La méthode ***htmlToObject*** permet de transformer la réponse d'un moteur en Objet **Resultat.**
La méthode ***htmlToXML*** permet de transformer la réponse d'un moteur de recherche en XML suivant la DTD décrite ci-dessous.

5) Fichiers connexes

a) Fichier XML de transfert inter-agents

Fichier Resultat.xml utilisé lors de la communication inter-agent (Doctype et court exemple) :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
<!DOCTYPE search [
  <!ELEMENT search (keywords,list)>
  <!ELEMENT keywords (#PCDATA)>
  <!ELEMENT list (link)*>
  <!ELEMENT link (title, url, desc)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT url (#PCDATA)>
  <!ELEMENT desc (#PCDATA)>
  <!ATTLIST list links CDATA #IMPLIED>
  <!ATTLIST link rank CDATA #IMPLIED>
  <!ATTLIST link score CDATA #IMPLIED>
]>
<search>
      <keywords>Exemples de recherche</keywords>
                  <list links="2">
                        <link rank="1" score="0">
                              <title>INRA - Exemples de recherche</title>

      <url>http://www.inra.fr/les_recherches/exemples_de_recherche</url>
                              <desc>Faits marquants de recherche, exemples
d'activités.</desc>
                        </link>
                        <link rank="2" score="0">
                              <title>INRA - Premier séquençage d'une bactérie du
yaourt : Streptococcus ...</title>
                              <url>http://www.inra.fr/les_recherches/e</url>
                              <desc>La recherche Inra : pourquoi, sur quoi,
comment ? | Exemples de recherche | Annuaire des sites Web | Ressources
scientifiques ...</desc>
                        </link>

                  </list>
</search>
```

b) Fichier de configuration d'un agent

Fichier metamoteur.conf permettant la configuration d'un agent par un utilisateur confirmé.

```
# Fichier de configuration du méta-moteur

# Pour les informations de connexion sur la base de données
# HostBDD contient aussi le port de connexion exemple localhost:3306
```

```
HostBDD
UserBDD
PassBDD
BaseBDD

# Pour les informations concernant les messages internes de l'agent
# Debug   0 : aucun message
#         1 : FichierLog
#         2 : Sortie standard
#         3 : FichierLog + Sortie Standard
Debug

# emplacement du fichier Log si Debug 1 ou 3
# FichierLog

# Les timeout en milliseconde
# TimeOutAgent : temps de contact maximum inter-agents
# TimeOutMoteurRecherche : temps de contact maximum moteurs de recherche
# TimeOutLocal : temps de réponse interne (supérieur aux 2 autres TimeOut
additionnés )
TimeOutAgent
TimeOutMoteurRecherche
TimeOutLocal

# Configuration pour le serveur
PortServeur 8080
NbThread 5
PageIndex index.html

# Configuration pour les autres agents
# exemple : Contacts 192.168.1.45:8080 82.232.245.248.5:5001
Contacts
```