

UNIVERSITÉ DE MONTRÉAL

UN MODÈLE DE RECHERCHE
D'INFORMATION SUR LE WEB INVISIBLE

FABIEN-KENZO SATO
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES (M.Sc.A.)
(GÉNIE INFORMATIQUE)
AVRIL 2003



National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisitiions et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-89220-4

Our file Notre référence

ISBN: 0-612-89220-4

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this dissertation.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de ce manuscrit.

While these forms may be included in the document page count, their removal does not represent any loss of content from the dissertation.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

UN MODÈLE DE RECHERCHE
D'INFORMATION SUR LE WEB INVISIBLE

présenté par: SATO Fabien-Kenzo

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. QUINTERO, Alejandro, Ph.D., président

M. PIERRE, Samuel, Ph.D., membre et directeur de recherche

M. GLITHO, Roch, Ph.D., membre et codirecteur de recherche

M. GUIBAULT, François, Ph.D., membre

REMERCIEMENTS

Je voudrais exprimer mes remerciements à mon directeur de recherche, le professeur Samuel Pierre, pour sa disponibilité, son support et son encadrement durant mon cursus de maîtrise.

L'expression de ma reconnaissance va également à l'endroit de mon codirecteur de recherche, Roch Glitho, pour son support et ses commentaires lors de mon passage à Ericsson. Mes remerciements vont ensuite à tous les membres du LARIM avec lesquels j'ai partagé des moments de travail mais aussi de détente.

Enfin, si ce mémoire a pu aboutir, c'est aussi grâce au soutien de ma famille et de mes amis qu'ils soient ici à Montréal ou sur d'autres continents, et à qui je désire exprimer ma gratitude pour leurs encouragements et leur support. En particulier, merci à Violaine, Laurent et Michael.

RÉSUMÉ

Devant la taille immense acquise par le Web, la recherche d'information sur le Web est devenue un enjeu extrêmement important. De nombreux outils très performants comme les moteurs de recherche existent, mais ils ne permettent pas d'effectuer des recherches de manière simple sur le Web dit *invisible*.

L'objet du présent mémoire est l'élaboration d'un modèle de recherche sur le Web invisible en utilisant le paradigme d'*agent mobile*. L'architecture développée utilise la puissance d'un moteur de recherche pour lui fournir une liste de sites du Web invisible susceptibles d'être intéressants, et lance une recherche dynamique sur ces sites grâce à des agents mobiles.

En plus de la version utilisant des agents mobiles, pour des fins de comparaison et d'expérimentation en conditions réelles, nous avons implémenté une version utilisant le paradigme traditionnel client/serveur. Nous avons ainsi pu tester la version client/serveur sur des sites Web réels, et les résultats qualitatifs obtenus sont très satisfaisants.

Nous avons réalisé, à l'aide d'un site test, des séries d'expériences comparatives des deux versions implémentées. Il en ressort que la version utilisant des agents mobiles génère beaucoup moins de trafic et est ainsi plus rapide que la version client/serveur, surtout lorsque le débit de la connexion est faible. De plus, comme les agents mobiles effectuent les calculs sur le serveur et non sur le client, cette approche soulage les ressources du terminal client. L'approche agent/mobile semble donc particulièrement avantageuse dans le cas de terminaux aux ressources faibles de type PDA.

Notre travail a permis de développer un modèle de recherche d'information sur le Web invisible à base d'agents mobiles et de montrer expérimentalement leurs avantages théoriques attendus.

ABSTRACT

In front of the huge size acquired by the Web, information retrieval on the Web became an extremely important stake. Many powerful tools like search engines exist, but they do not make it possible to carry out research in a simple way on the part of the Web known as the *invisible Web*.

The purpose of the present thesis is the development of a model of information retrieval on the invisible Web by using the paradigm of *mobile agent*. The architecture developed uses the power of a search engine to provide a list of sites of the invisible Web likely to be interesting, and launches a dynamic research on these sites thanks to mobile agents.

To compare and experiment in real conditions, we implemented a version using traditional paradigm client/server in addition to the version using mobile agents. We thus could test the client/server version on actual Web sites, and the qualitative results obtained are very satisfactory.

We carried out, using a test site, series of comparative experiments of the two versions implemented. This reveals that the version using mobile agents generates much less traffic and thus is faster than the client/server version, especially when the flow of connection is weak. Moreover, as the mobile agents carry out calculations on the server and not on the client, this approach relieves the resources of the client terminal. Thus the mobile agent approach seems particularly advantageous in the case of weak resource terminals like PDA.

Our work made it possible to develop a model of information retrieval on the invisible Web based on mobile agents, and it showed in experiments their awaited theoretical advantages.

TABLES DES MATIÈRES

REMERCIEMENTS	iv
RÉSUMÉ	v
ABSTRACT	vi
TABLES DES MATIÈRES	vii
LISTE DES TABLEAUX	x
LISTE DES FIGURES	xi
LISTE DES SIGLES ET ABRÉVIATIONS	xiii
CHAPITRE I INTRODUCTION	1
1.1 Définitions et concepts de base	1
1.2 Éléments de la problématique	3
1.3 Objectifs de recherche	4
1.4 Contributions des travaux	5
1.5 Plan du mémoire	5
CHAPITRE II LA RECHERCHE D'INFORMATION SUR LE WEB	6
2.1 Les outils de recherche	6
2.1.1 Les annuaires	6
2.1.2 Les moteurs de recherche	7
2.1.3 Les méta-moteurs	8
2.1.4 Les autres outils de recherche	9
2.2 Les limites et les défis des moteurs de recherche	11
2.2.1 Le « crawling »	11
2.2.2 Le stockage	14
2.2.3 Les algorithmes de pertinence	14
2.3 Le Web invisible	16
2.3.1 Raison d'être du Web invisible	16
2.3.2 Classification du Web invisible	20
2.3.3 Les outils de recherche sur le Web invisible	22

2.3.4 Les évolutions futures du Web invisible.....	23
2.4 Les agents mobiles et la recherche d'information.....	25
2.4.1 Le concept d'agent mobile.....	25
2.4.2 Les avantages et limitations des agents mobiles.....	27
2.4.3 Les agents mobiles dans la recherche d'information.....	30
CHAPITRE III RECHERCHE DYNAMIQUE SUR LE WEB INVISIBLE.....	32
3.1 Architecture générale.....	33
3.2 Recherche à partir d'un serveur dédié.....	34
3.2.1 Approche client/serveur.....	35
3.2.2 Approche par agents mobiles.....	35
3.2.3 Avantages et inconvénients des deux approches.....	36
3.3 Recherche à partir du terminal de l'utilisateur.....	37
3.3.1 Approche client/serveur.....	37
3.3.2 Approche par agents mobiles.....	38
3.3.3 Avantages et inconvénients des deux approches.....	40
3.4 Intérêts et limites.....	40
3.6 Améliorations possibles.....	42
CHAPITRE IV IMPLÉMENTATION ET MISE EN ŒUVRE.....	44
4.1 Environnement de réalisation.....	44
4.1.1 La plate-forme d'agent mobile Grasshopper.....	44
4.1.2 Le langage de programmation Java.....	45
4.1.3 Le système d'exploitation.....	45
4.2 Choix d'implémentation.....	45
4.2.1 Délimitation de la partie à implémenter.....	46
4.2.2 Type de formulaire.....	47
4.2.3 Type d'expressions prises en compte.....	49
4.2.4 Profondeur de recherche.....	49
4.2.5 Résultats de la recherche.....	50
4.3 Implémentation de l'approche client/serveur.....	52

4.3.1 La bibliothèque HTMLParser	52
4.3.2 Le package <i>Dynamix</i>	55
4.4 Implémentation de l'approche agent mobile	57
4.4.1 Structure de l'agent mobile <i>DynamixMA</i>	58
4.4.2 Exécution	60
CHAPITRE V EXPÉRIMENTATION ET RÉSULTATS	61
5.1 Tests qualitatifs	61
5.2 Configuration des machines tests	63
5.2.1 Configuration du serveur	63
5.2.2 Configuration du client	64
5.2.3 Site test	64
5.3 Expériences	66
5.3.1 Évaluation de la réponse en temps avec une connexion faible débit	66
5.3.2 Évaluation de la réponse en temps avec une connexion haut débit	70
5.3.3 Évaluation du trafic généré	74
5.3.4 Évaluation de la consommation des ressources des machines	78
5.4 Synthèse des performances	81
CHAPITRE IV CONCLUSION	83
6.1 Synthèse des travaux et contributions	83
6.2 Limitations des travaux	84
6.3 Orientations de recherches futures	85
BIBLIOGRAPHIE	87

LISTE DES TABLEAUX

Tableau 5.1 Taille des pages du site test	65
Tableau 5.2 Réponse en temps de la version client/serveur faible débit (56 Kbps)	67
Tableau 5.3 Réponse en temps de la version agent mobile faible débit (56 Kbps)	69

LISTE DES FIGURES

Figure 1.1 Architecture d'un moteur de recherche	2
Figure 2.1 Trafic généré par les deux approches	28
Figure 3.1 Architecture avec recherche dynamique à partir d'un serveur dédié	34
Figure 3.2 Architecture avec recherche dynamique à partir de la machine utilisateur	38
Figure 3.3 Parcours des agents mobiles	39
Figure 4.1 Sous-ensemble implémenté	46
Figure 4.2 Schéma général de l'application	47
Figure 4.3 Formulaire sur la page d'accueil du site du quotidien Libération	48
Figure 4.4 Page web compilant les résultats d'une recherche	51
Figure 4.5 La classe HTMLParser	53
Figure 4.6 La classe HTMLFormTag	54
Figure 4.7 Le package Dynamix	54
Figure 4.8 Code HTML d'un formulaire	55
Figure 4.9 Un exemple de formulaire	56
Figure 4.10 Code simplifié de l'agent mobile DynamixMA	59
Figure 4.11 Interface graphique de Grasshopper	60
Figure 5.1 Un exemple de recherche.....	62
Figure 5.2 Réponse en temps de la version client/serveur faible débit (56 Kbps).....	68
Figure 5.3 Réponse en temps de la version agent mobile faible débit (56 Kbps).....	68
Figure 5.4 Comparaison des temps de réponse (faible débit)	70
Figure 5.5 Réponse en temps de la version client/serveur haut débit (100 Mbps)	71
Figure 5.6 Réponse en temps de la version agent mobile haut débit (100 Mbps)	73
Figure 5.7 Comparaison des temps de réponse (haut débit)	73
Figure 5.8 Trafic généré par l'approche client/serveur.....	75
Figure 5.9 Trafic généré par l'approche agent mobile.....	77

Figure 5.10 Comparaison des trafics générés	78
Figure 5.11 Utilisation du processeur dans l'approche client/serveur	80
Figure 5.12 Utilisation du processeur dans l'approche agent mobile	80

LISTE DES SIGLES ET ABRÉVIATIONS

ADSL	Asymmetric Digital Subscriber Line
API	Application Programming Interface
ASP	Active Servers Pages
CGI	Common Gateway Interface
FIPA	Foundation for Intelligent Physical Agent
HITS	Hypertext Induced Topic Search
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfert Protocol
IP	Internet Protocol
JSP	Java Server Pages
Kbps	Kilobits par seconde
Ko	Kilooctets
MASIF	Mobile Agent System Interoperability Facility
Mbps	Megabits par seconde
Mo	Mégaoctets
NASA	National Aeronautics and Space Administration
OMG	Object Management Group
PDA	Personal Digital Assistant
PDF	Portable Document Format
PHP	PHP : Hypertext Preprocessor
RPC	Remote Procedure Call
TCP	Transmission Control Protocol
To	Téraoctets
UML	Unified Modeling Language
XML	eXtensible Markup Language

CHAPITRE I

INTRODUCTION

Aujourd'hui, pour une part sans cesse croissante de la population mondiale, l'Internet, et tout particulièrement avec le Web, est devenu un outil d'information privilégié voire indispensable. En effet, l'Internet permet, aussi bien à des entreprises ou organismes qu'à des particuliers, de publier de l'information à l'échelle mondiale et ceci, à très faible coût. L'information disponible sur le Web continue de s'accroître de façon exponentielle, et il devient de plus en plus difficile de s'orienter dans une telle masse d'informations très peu structurée. Il est ainsi devenu nécessaire de mettre à la disposition des utilisateurs du Web des outils de recherche adaptés, et les moteurs de recherche sont désormais essentiels à la consultation du Web. Cependant, le Web a évolué et il est maintenant constitué d'une grande part de sites dynamiques qui prodiguent une information abondante et de qualité mais qui reste encore inaccessible aux moteurs de recherche. L'objet de ce mémoire est la mise au point d'une solution permettant de rendre cet accès possible. Dans ce chapitre d'introduction, nous présentons d'abord quelques concepts de base et des éléments de la problématique, avant de résumer nos objectifs de recherche, les résultats escomptés, la contribution de nos travaux, et le plan du mémoire.

1.1 Définitions et concepts de base

Un *moteur de recherche* est un outil informatique, entièrement automatisé, qui génère une liste de liens vers des pages Web correspondant à la requête de l'utilisateur. En général, il possède une interface Web et est donc consultable directement sur le Web. Il convient de distinguer les moteurs de recherche des *annuaires* qui classent hiérarchiquement les sites Web dans un répertoire selon une évaluation «humaine», alors que les moteurs de recherche agissent de manière entièrement automatique.

Les moteurs de recherche sont constitués de trois composantes principales [1] : le *robot*, l'*index* et le *module de requêtes* (Figure 1.1).

1. Le robot, appelé aussi *crawler* ou *spider*, parcourt continuellement le Web en rapatriant le contenu des pages Web et en suivant les liens hypertexte vers d'autres pages.
2. L'index extrait les mots de chaque page et enregistre l'URL correspondante. L'information est structurée dans des bases de données immenses selon des logiques propres à chaque moteur de recherche.
3. Le module de requêtes reçoit les requêtes des usagers, cherche dans l'index, classe et retourne les résultats.

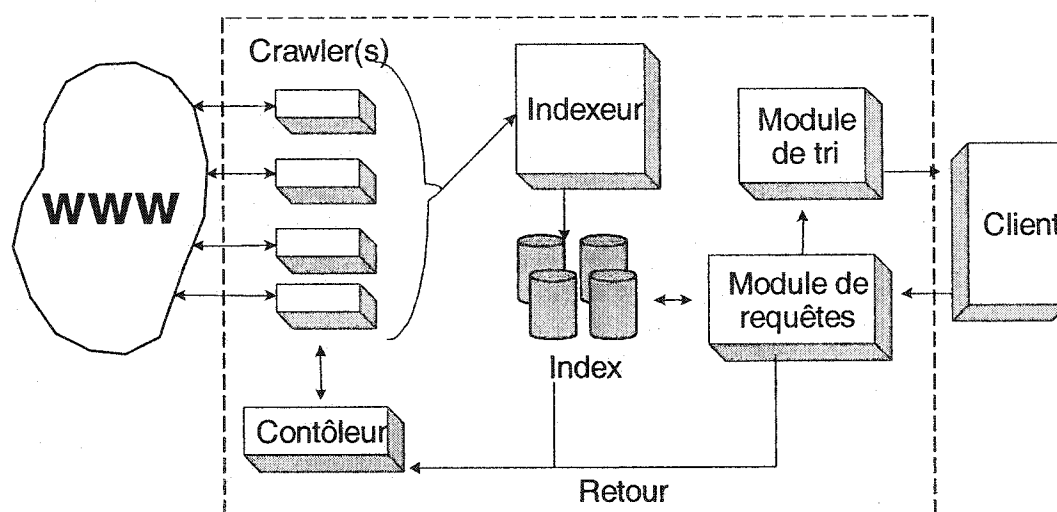


Figure 1.1 Architecture d'un moteur de recherche

Un *site Web* est dit dynamique lorsque ses pages ne sont générées qu'à la demande, contrairement aux sites statiques pour lesquels chacune de leurs pages préexiste sur le serveur web. Les premiers sites Web étaient constitués d'une arborescence fixe et finie de pages HTML. Ce type de conception est simple et clair. Néanmoins, les choses se compliquent lorsque le site est de très grande taille, lorsqu'il nécessite une mise à jour fréquente, ou lorsque l'on veut personnaliser les pages en fonction du profil du client. C'est pourquoi, de nouvelles technologies permettant de

rendre les sites *dynamiques* ont émergé. Elles reposent sur l'utilisation de bases de données pour stocker et gérer facilement un grand nombre de données. Elles créent une interface Web à ces bases, en générant automatiquement des pages Web, en fonction de pages Web modèles, de la requête du client, et de la réponse fournie par la base de données. Ces pages Web n'existent donc pas sur le serveur avant la requête du client. Les sites Web dynamiques permettent de bénéficier des avantages des bases de données, c'est à dire la gestion de grandes quantités de données et la possibilité d'effectuer des requêtes complexes.

1.2 Éléments de la problématique

À l'heure actuelle, la majeure partie des données accessibles publiquement sur le Web, n'est pas indexé par les moteurs de recherche. Cette partie du Web est appelée le *Web invisible*. Ces pages Web ne sont pas indexées pour des raisons diverses :

- Les pages sont isolées, aucun lien ne pointe vers elles. La seule chose à faire est une soumission manuelle.
- Les pages sont trop profondes dans la hiérarchie du site. Les moteurs de recherche choisissent souvent de ne pas les indexer.
- Les pages sont trop récentes ou sont remises à jour trop fréquemment pour que les moteurs de recherche les indexent. Les crawlers mettent en effet plusieurs semaines à parcourir le Web.
- L'accès du site est interdit aux crawlers par le concepteur du site. Il est en effet possible de réguler leur accès dans un fichier *robot.txt*.
- Les fichiers sont de format particulier, comme par exemple des fichiers multimédia. Il est à noter que des moteurs de recherche, et en particulier *Google*, indexent un nombre sans cesse croissant de types de fichiers.
- Le site est dynamique. Le crawler ne peut pas accéder aux pages car elles n'existent pas sur le serveur, et il ne peut les générer car il n'a pas de requêtes à faire.

Nous nous limitons au dernier point. En effet, les sites dynamiques représentent désormais la majeure part de l'information disponible sur le Web et leur nombre croît beaucoup plus vite que celui des sites traditionnels. La plupart des grands organismes, comme des centres de recherche ou des bibliothèques, mettent leurs bases de données en ligne aux moyens de sites Web dynamiques. Ces sites abondent d'information spécialisée et de qualité.

Le site *BrightPlanet* a réalisé en 2001 une étude [2] sur le Web invisible (Nous assimilerons, pour toute la suite, le Web invisible aux sites Web dynamiques). Il en ressort notamment que :

- le Web visible représente 19 téraoctets (To) contre 7500 To pour le Web invisible;
- 95% du Web invisible est accessible publiquement, il n'y a pas d'inscription ni de frais;
- les sites du Web invisible sont plus fortement maillés que ceux du Web visible.

Nous comprenons alors qu'il y a un besoin à combler concernant la recherche sur le Web invisible.

1.3 Objectifs de recherche

Le but de nos travaux est d'ajouter une fonctionnalité aux moteurs de recherche leur permettant d'accéder aux sites dynamiques. Nous désirons mettre au point une architecture à partir de moteurs de recherche traditionnels qui effectuerait une recherche complémentaire à partir des résultats obtenus lors de la recherche classique.

La première étape consiste à indexer les liens vers des sites dynamiques contenus dans les pages classiques parcourues par le crawler.

La seconde étape consiste à accéder aux sites Web dynamiques repérés lors de la recherche classique en faisant une requête appropriée. Nous souhaitons comparer deux méthodes : une approche traditionnelle client/serveur et une approche utilisant les agents mobiles. Un des objectifs de ces travaux étant de motiver l'utilisation des agents mobiles.

1.4 Contributions des travaux

La principale contribution de ce mémoire sera le développement d'une architecture permettant de réaliser des recherches sur des sites Web dynamiques. Un avantage important de notre méthode est que le contenu des sites n'est pas indexé, ce qui évite des problèmes de stockage. De plus, l'information fournie est fraîche, il n'y a pas de liens morts.

Un élément important de nos travaux sera l'implémentation du modèle suivant les approches client/serveur et agents mobiles, et la comparaison de ces deux approches. Nos travaux permettront de développer et d'évaluer les performances d'une application pour agents mobiles.

1.5 Plan du mémoire

Le deuxième chapitre de ce mémoire fait l'état de l'art du domaine de la recherche d'information sur le Web et explicite les concepts de Web invisible et d'agent mobile. Le troisième chapitre présente l'architecture de notre modèle de recherche sur le Web invisible. L'implémentation de ce modèle selon deux approches différentes client/serveur et par agent mobile fera l'objet du quatrième chapitre. Dans le cinquième chapitre, nous détaillerons les tests réalisés afin de montrer la pertinence de notre modèle et de comparer les deux versions implémentées, et nous analyserons les résultats obtenus. Enfin, le dernier chapitre conclura ce mémoire en faisant la synthèse de nos travaux, en présentant leurs limitations et en indiquant nos recherches futures.

CHAPITRE II

LA RECHERCHE D'INFORMATION SUR LE WEB

Internet et le Web en particulier sont devenus la plus grande source d'information au monde. De plus, elle est accessible très facilement depuis de nombreux points de la planète. De l'information aussi bien générale, d'actualité ou bien spécialisée y est présente. Il est nécessaire de s'orienter dans cette masse de données peu structurée, et aujourd'hui il existe de nombreux outils de recherche, dont les plus puissants sont les moteurs de recherche, qui fonctionnent très bien. Cependant, ils ne peuvent couvrir tout le Web. Cette partie cachée du Web est appelée *Web invisible*. Nous allons donc étudier les différents outils de recherche et leurs limites, puis nous analyserons le Web invisible. Nous finirons par introduire le concept d'*agent mobile*, et son rôle dans la recherche d'information.

2.1 Les outils de recherche

Il existe différents outils de recherche dont les principes de fonctionnement et les rôles sont très variés. Il convient de bien choisir son outil de recherche, en fonction de ses besoins. La lecture de l'article de François Bourdoncle [3] et la consultation du site Web www.abondance.com [4] ont été très utiles pour la rédaction de cette section.

2.1.1 Les annuaires

Un annuaire est un outil de recherche qui répertorie des sites Web. Chaque site Web référencé est classé dans une ou plusieurs catégories et est décrit brièvement. On peut ainsi faire une recherche arborescente par catégories ou bien une recherche par mots-clés, l'annuaire effectue alors une recherche dans les mots contenus dans ses fiches descriptives. Il renvoie ainsi une liste de sites, leur description et leur adresse.

Un annuaire référence donc des sites et non leur contenu. On obtient l'adresse de la page d'accueil des sites. Ainsi, un annuaire est pertinent lorsque l'on recherche des sites de référence dans un domaine particulier ou bien des sites d'informations commerciales. Mais il ne pourra effectuer une recherche à l'intérieur même des sites Web qu'il référence. Il n'est donc pas adapté à des recherches très pointues.

Les annuaires utilisent un référencement manuel. L'avantage est que l'on aura une description cohérente du site, contrairement à un référencement automatisé qui peut se laisser piéger par des balises HTML ou faire des erreurs de compréhension syntaxique. L'inconvénient est que l'on ne peut pas référencer autant de sites. C'est pour cette raison que la plupart des annuaires sont maintenant associés à des moteurs de recherche, comme par exemple Yahoo avec Google. Cela permet de lancer des recherches complémentaires lorsque leur index ne suffit pas.

Il existe de nombreux annuaires généralistes et spécialisés. Il existe même des annuaires du Web invisible.

2.1.2 Les moteurs de recherche

Un moteur de recherche [1], bien que d'aspect semblable lors de sa consultation, est un outil fonctionnant sur un principe totalement différent de celui de l'annuaire. Alors que le référencement des sites se fait manuellement dans le cas de l'annuaire, il est entièrement automatisé pour le moteur de recherche.

Des robots logiciels appelés « crawler » ou « spider » parcourent le Web de liens en liens. Ils rapatrient les pages ainsi visitées, et leur contenu est stocké dans un index structuré qui peut atteindre des tailles colossales. Lorsqu'un utilisateur effectue une requête, le moteur consulte son index, en extrait les réponses correspondantes et les classe dans un ordre le plus pertinent possible suivant un algorithme spécifique.

La procédure entièrement automatisée permet de référencer bien plus d'information que les annuaires. Et dans le cas des moteurs, ce n'est pas le site qui est référencé, mais tout son contenu. L'usage des moteurs de recherche est particulièrement

adapté lorsque l'on désire effectuer une recherche très précise. Cependant, comme la quantité d'information indexée est gigantesque, on risque d'avoir plusieurs dizaines de milliers de réponses. Bien que les différents moteurs de recherche s'efforcent d'améliorer leur algorithme de tri afin de faire apparaître les résultats les plus pertinents en premier, ils ne peuvent pas toujours lire nos pensées. Il convient donc de formuler notre requête le plus précisément. C'est pourquoi tous les moteurs de recherche permettent d'effectuer une recherche avancée. On peut utiliser des opérateurs booléens, l'opérateur de troncature, faire une recherche suivant une expression complète, ou encore, limiter notre recherche à un domaine particulier du Web (par exemple « .ca », si l'on veut se limiter aux sites canadiens). Tous les moteurs de recherche n'offrent pas les mêmes fonctionnalités, ni la même syntaxe.

2.1.3 Les méta-moteurs

Un méta-moteur est un outil qui interroge en parallèle plusieurs moteurs de recherche classiques et/ou des annuaires. Il rapatrie ensuite les résultats des différentes requêtes et les fusionne de la manière la plus pertinente possible. Il existe deux formes de méta-moteurs : les méta-moteurs logiciels qu'on installe sur sa machine comme Copernic ou Webseeker, et les méta-moteurs Web que l'on interroge via un site Web comme Kartoo ou Metacrawler.

Nous pourrions penser fort logiquement que les méta-moteurs donnent des meilleurs résultats que les moteurs classiques. Il est en réalité très difficile de réaliser, de façon pertinente, la synthèse des résultats issues des différents moteurs sollicités qui utilisent tous des techniques de classement distinctes. De plus, il n'est pas possible d'accéder via les méta-moteurs aux fonctionnalités de recherche avancée disponibles sur les moteurs. Il est donc difficile de réaliser des recherches poussées. On obtient généralement de meilleurs résultats en exploitant correctement les ressources d'un seul moteur de recherche, c'est à dire en formulant des requêtes précises et en utilisant les opérateurs booléens ou autres fonctionnalités disponibles. Enfin, d'un point de vue légal,

nous pouvons nous poser la question : peut-on exploiter sans contreparties les moteurs de recherches disponibles sur le Web afin de s'en approprier les résultats à des fins commerciales?

Les méta-moteurs qui apparaissaient comme une solution très intéressante n'ont toujours pas convaincu les utilisateurs.

2.1.4 Les autres outils de recherche

À côté des annuaires et des moteurs de recherche qui sont, de loin, les outils de recherche les plus utilisés sur le Web, d'autres outils existent [4]. Certains sont plus spécialisés, d'autres plus innovants. Leur caractéristique commune est qu'ils sont encore marginaux, du moins du point de vue du grand public.

Les agents de veille

Plusieurs sites Web (Netmind, Spyonit, Get Updated, ...) proposent des services de veille. Ils évitent ainsi d'effectuer des recherches répétitives lorsque l'on guette des innovations dans un secteur de recherche ou d'intérêt.

Ces agents de veille offrent deux services principaux. Ils vont analyser des pages Web dont on leur a donné l'adresse, les comparent à leur dernière version et nous avisent des changements. Ils exécutent également des requêtes avec des mots-clés qu'on leur fournit sur des outils de recherche, et nous préviennent si de nouveaux résultats apparaissent. L'intervalle de recherche est en général configurable.

Ce service permet un gain de temps précieux, mais il est limité à un type de recherche très précis.

Les outils de recherche humains

Ce type de service est basé sur des interlocuteurs humains qui vont tenter de répondre à notre requête. En général, on pose notre question sur le site Web et on obtient la réponse, soit plus tard par courriel, soit en direct sur le site. Un grand nombre d'outils

de recherche humains fleurissent sur la toile¹. Après les sites spécialisés, des poids lourds de la recherche d'information comme Yahoo ou Google lancent leur service de recherche humain. Beaucoup de services sont payants.

On distingue deux types de service. Le premier est semblable à un centre d'appels, où les opérateurs sont des professionnels de la recherche d'information. Ils vont donc effectuer une recherche à notre place et ceci assez rapidement. Le site le plus populaire est WebHelp. Cette option est intéressante pour les internautes qui ne sont pas très familiers avec la recherche d'information, et qui ne désirent pas y passer trop de temps.

Le deuxième type de service offre des réponses d'experts. Certains sites sont généraux, comme Question.fr, d'autres comme ExpertCity, sont spécialisés. Ce service est particulièrement utile lorsque l'on désire une information très spécialisée. Le site envoie alors la question à des experts dans le domaine en question, et l'on reçoit la réponse plus tard.

Le Peer To Peer

De nombreux logiciels « Peer to Peer » permettent l'échanges de données sur Internet entre utilisateurs en permettant le partage de portions de disques durs. Le plus connu de ces logiciels est Napster qui permettent l'échange de fichiers musicaux au format MP3, mais aujourd'hui des logiciels plus récents(Kazaa, eDonkey) permettent de gérer un très grand nombre de formats de fichier.

Des concepteurs de moteurs de recherche dont Altavista ont étudié un système « Peer To Peer » adapté à la recherche d'information, qui en plus de la recherche classique sur leur index, la prolonge sur les portions de disques durs partagées par les internautes. Dans l'intranet d'une entreprise, ce système permettrait d'étendre une

¹ Une liste bien fournie est disponible sur le site Web Netsurf à l'adresse suivante:
<http://www.netsurf.ch/askexperts.html>

recherche à des présentations de format PowerPoint ou à des tableurs, voire à des e-mails. Il rendrait possible un meilleur partage de l'information.

Ce système pose cependant de sérieux problèmes de sécurité et de confidentialité. Il est cependant trop tôt pour juger de l'avenir de cette technologie dans le cadre de la recherche d'information.

Exalead

Exalead [5] (www.exalead.com) est un outil de recherche intelligent et innovant développé par une équipe française. Cette technologie s'intègre dans des portails ou des bases de données préexistants comme un annuaire Web et propose des fonctionnalités de recherche intelligentes. La plate-forme Exalead analyse statistiquement le contenu de la base de données et extrait des catégories et des mots-clés pertinents. Elle permet d'affiner une recherche par un simple clic sur la rubrique ou le mot-clé qui correspond le plus à l'objet de notre requête. Une démonstration est disponible sur le site. Exalead est un outil qui permet de trier intelligemment et rapidement les résultats d'une recherche. C'est assurément une technologie prometteuse.

2.2 Les limites et les défis des moteurs de recherche

Chaque outil de recherche possède ses spécificités et est adapté à un certain type de recherche. Il convient donc de bien choisir l'outil de recherche à utiliser. Cependant, les moteurs de recherche sont actuellement les outils les plus complets et les plus puissants. Nous allons ainsi nous concentrer sur cet outil et analyser les différents points qui limitent ses performances.

2.2.1 Le « crawling »

Si l'on ne considère que le *Web visible*, c'est à dire la portion du Web potentiellement indexable par les moteurs de recherche, nous pouvons donner quelques

estimations issues de différentes études [6,7] sur la taille du Web. Par ailleurs, nous reviendrons plus longuement sur la notion de Web visible. En 2001, il y avait au moins 2 millions de sites Web anglophones et 150 000 francophones [2]. Le nombre de pages Web étaient d'au moins 2 milliards toutes langues confondues et d'environ 20 millions pour celles en français. Google a passé le cap des 3 milliards de pages indexées fin 2002.

Face à cet océan de données, se pose le problème de son parcours ou encore du *crawling*. Les crawlers ne pouvant pas parcourir toutes les pages Web, il est nécessaire de faire des choix. Il faut donc estimer quelles pages sont prioritaires, afin de parcourir la portion de Web la plus significative possible. Tous les moteurs de recherche n'ont pas exactement les mêmes critères, ce qui explique qu'ils n'ont pas le même index et donc ne donneront pas le même résultat pour une requête identique. Certains critères peuvent être facilement exploités directement d'après l'adresse de la page. Un moteur peut privilégier certains domaines, un moteur de recherche francophone ira en priorité sur les sites des domaines .fr, .be ou .ca par rapport aux domaines .de ou .es par exemple. Les adresses contenant les termes «index» ou «home» sont avantagées car elles identifient souvent des pages d'accueil. Les pages ayant des adresses longues contenant de nombreux «/» sont pénalisées, étant considérées comme en général moins intéressantes que les pages moins profondes. Un autre critère plus élaboré et certainement plus significatif qui est essentiel dans la stratégie du moteur de recherche Google, est la *popularité* de la page. La popularité d'une page est estimée en tenant compte du nombre de liens qui pointent sur cette page. On peut affiner ce critère en considérant qu'un lien provenant d'une page « populaire » a plus d'importance qu'un lien provenant d'une page quelconque. Ceci a l'avantage de pénaliser les pages qui essaient de se faire bien référencer de façon « sauvage ». Il est en effet possible de créer des liens vers des pages de son choix, grâce à des pages de liens dont le seul but est d'augmenter les chances de référencer des sites.

Une fois que les crawlers ont choisi et parcouru les pages importantes, il faut rafraîchir l'index des pages téléchargées. En effet, ces dernières peuvent être mises à

jour, ou tout simplement disparaître. De plus, de nouvelles pages apparaissent sans cesse. Le Web est en effet un support où l'information est extrêmement dynamique, et il évolue très rapidement. Les conséquences d'un rafraîchissement trop peu fréquent peuvent se révéler désastreuses : des liens pointant vers des pages qui ne contiennent plus l'information recherchée; des liens « morts », c'est à dire pointant vers des pages qui n'existent plus; et l'absence de pages récentes, pourtant très pertinentes. Il convient donc d'optimiser le rafraîchissement de l'index. Naïvement, il suffirait de le rafraîchir très souvent, une fois par jour par exemple. Mais on se heurte à des limites techniques, dues au trop grand volume d'information indexée. Le délai de rafraîchissement de l'index va en effet de trois semaines à deux mois, suivant le moteur de recherche considéré. Il est nécessaire de souligner l'hétérogénéité des mises à jour des pages web. Alors que des sites d'actualité sont remis à jour quotidiennement voire plusieurs fois par jour, des sites d'archive ne changent pratiquement jamais. Il est donc judicieux d'adapter la fréquence de rafraîchissement aux sites considérés. Un site qui ne change presque pas, mais également un site qui change trop souvent comme un site d'informations en temps réel permettant par exemple de suivre un événement sportif minute par minute, n'ont pas besoin d'être rafraîchis. Pour le reste des sites, il faut adapter le rafraîchissement à leur probabilité de mise à jour. Il existe plusieurs stratégies basées sur des études statistiques [8,9].

Afin d'optimiser les ressources et le temps de parcours du Web, plusieurs crawlers sont utilisés en parallèle. Ceci crée un besoin de coordination, afin de réaliser un travail global efficace. Les différents crawlers, par exemple, ne doivent pas visiter le même site. On ne peut pas non plus multiplier indéfiniment le nombre de crawlers, car cela créerait une surcharge de communication significative. Les crawlers sont coordonnés par un module de contrôle qui leur assigne la liste des sites qu'ils doivent visiter [10].

2.2.2 Le stockage

En considérant la taille des données à stocker dans l'index, le stockage est un problème très important dans la conception des moteurs de recherche. Cependant ce point n'est pas spécifique aux moteurs de recherche, et il constitue le domaine de recherche bien connu des bases de données. Nous nous contentons donc de citer les points importants du stockage dans ce contexte précis.

Le dépôt de données doit être distribué et souple afin de pouvoir l'agrandir simplement en ajoutant des disques durs et des ordinateurs, et afin d'être réparti au mieux sur l'Internet. Un moteur de recherche comme Google localise ses principaux serveurs aux États-Unis mais installe des serveurs miroirs au Royaume-Uni et des serveurs gérant les pages francophones en France. Cette distribution entraîne évidemment des problèmes de cohérence. Le dépôt de données doit aussi supporter deux types d'accès. Le « random access » utilisé pour accéder à une page spécifique et le « streaming access » pour recevoir du crawler une collection complète de pages. De plus, il doit gérer efficacement des mises à jour importantes et continues en suivant le rythme des crawlers.

2.2.3 Les algorithmes de pertinence

Un des défis les plus importants des moteurs de recherche est le tri des résultats des requêtes. En effet, un moteur aurait beau indexer toute l'information disponible sur le Web, s'il n'est pas capable de retourner des résultats pertinents, il sera peu performant dans la recherche d'information. À l'heure actuelle, les moteurs sont très peu dotés d'« intelligence », ils agissent principalement en reconnaissant des chaînes de caractères et en comparant leur occurrence. Ils ne sont pas encore capables de raisonner sur la sémantique ou de reconnaître une idée. La polysémie est un élément particulièrement important. Suivant le contexte, un mot peut avoir des significations très différentes, mais les moteurs ne sont pas capables de faire la différence. De plus, une requête peut avoir plusieurs dizaines de milliers de résultats, et il est très difficile de déterminer quelles

pages sont les plus pertinentes par rapport à la requête de l'utilisateur. Et l'utilisateur va en général se contenter de visiter les premières pages parmi la liste des réponses retournées. On comprend alors l'importance de l'algorithme de tri, ou de pertinence, utilisé.

Les moteurs de recherche sont aujourd'hui très populaires et sont un des moyens principaux d'accès aux sites web. Les sites à vocation commerciale ont ainsi un très fort intérêt à se faire bien référencer. Pour les webmestres tous les moyens sont bons pour essayer d'arriver en bonne position lors des requêtes sur les moteurs. Ils insèrent par exemple du texte caché contenant des mots-clés parasites ou ont recours à trop de balises Meta. Les concepteurs de moteurs de recherche sont obligés de prendre des mesures afin de contrer ces « spammers ». Les algorithmes de pertinence deviennent donc de plus en plus complexes. De plus, comme ils diffèrent selon les moteurs utilisés, il est difficile de se faire bien référencer sur plusieurs moteurs à la fois.

Après les premiers algorithmes basés simplement sur la correspondance entre les mots de la requête et les mots contenus dans la page, une nouvelle génération d'algorithmes basés sur les liens hypertextes et la structure de graphe du Web a vu le jour. L'algorithme le plus connu est le « Page Rank » mis au point par les concepteurs de Google qui donne plus d'importance aux pages qui ont beaucoup de liens pointant sur elles, d'autant plus si les liens proviennent de pages considérées elles-mêmes comme importantes. Grâce notamment à cette idée, Google a conquis le marché des moteurs de recherche. Il existe d'autres algorithmes de ce type, la structure de graphe particulière du Web stimulant des recherches intéressantes sur le sujet. Par exemple, l'algorithme HITS (Hypertext Induced Topic Search) [11] distingue les pages « autorités » qui sont des pages offrant de l'information sur un sujet particulier, des pages « hub » qui pointent vers plusieurs pages « autorités ».

Cependant, la sémantique constitue le domaine de recherche le plus prometteur en matière d'algorithmes de pertinence. Exalead, que nous avons cité dans la partie précédente en est un bon exemple. Cette technologie permet en effet de raffiner une

recherche par mots-clés ou thèmes qui peuvent se rapprocher de la requête. Si on lance une recherche avec le mot-clé « Cléopâtre », on nous proposera les rubriques « histoire » et « cinéma » et des mots-clés comme « aventures d'Astérix », « l'Égypte des pharaons » ou « Caius Julius ». Les utilisateurs pourront affiner leur recherche suivant par exemple qu'ils recherchent des informations sur le film « Astérix et Obélix : Mission Cléopâtre » ou bien sur l'histoire de l'antiquité égyptienne.

2.3 Le Web invisible

Le Web visible représente l'ensemble des pages Web qui sont indexées par les moteurs de recherche. Le Web invisible [12] est son complémentaire. Cependant, il est très hétérogène, et les pages le constituant ne sont pas indexées pour des raisons très différentes les unes des autres. Ces raisons peuvent être des impossibilités technologiques, des problèmes de coûts, ou bien des choix stratégiques.

2.3.1 Raison d'être du Web invisible

Les pages isolées

Les crawlers des moteurs de recherche ne peuvent accéder aux pages Web que par l'intermédiaire de liens hypertextes. Ils ne peuvent donc pas accéder à des pages vers lesquelles aucun lien ne pointe. Le Web a la forme d'un graphe orienté dont les nœuds sont les pages Web et les arêtes les liens hypertextes. Ce graphe est essentiellement constitué d'un cœur fortement maillé, d'un ensemble de nœuds pointant vers le cœur, d'un autre pointé par le cœur et d'un certain nombre de pages isolées [13]. Pour qu'une page isolée soit indexée par un moteur de recherche, il faut soit qu'une autre page indexée pointe vers cette page et attendre le passage du crawler, soit soumettre l'URL de la page au moteur de recherche par l'intermédiaire de formulaires généralement disponible sur les sites des moteurs.

Les formats particuliers

Les moteurs de recherche ont été originellement conçus afin de trouver de l'information sous forme de texte. Cependant, le Web a considérablement évolué, et à l'heure actuelle, une part très importante du Web est constituée d'images, de vidéos, de sons, de flux multimédia (streaming), d'exécutables, ou bien encore de fichiers compressés (.tar, .zip, etc.). Bien que des moteurs de recherche spécialisés tentent d'indexer des images ou des sons, ils sont encore bien loin d'apporter des résultats satisfaisants et se contentent souvent d'utiliser la balise HTML décrivant l'objet.

Il convient de souligner l'importance de cette partie du Web, notamment les sites comprenant des éléments de type Flash. En effet, de très nombreux sites utilisent cette technologie permettant une interactivité, une animation et un graphisme très attractifs. Mais les crawlers n'y distinguent aucun texte et sont donc incapables de les indexer.

À côté de ces formats non textuels qui ne peuvent être indexés à cause d'impossibilités techniques, il existe des formats de texte élaborés tels le PDF ou le Postscript qui sont très souvent ignorés par les moteurs de recherche. Techniquement, il est facile de convertir des fichiers de tels formats en fichiers texte simples qui sont alors indexables. Cependant, la conversion est coûteuse en ressources, d'autant plus que les fichiers de ce type contiennent souvent plusieurs dizaines voire centaines de pages, et les moteurs de recherche faisaient le choix de les ignorer. Mais aujourd'hui, la plupart des sites administratifs, universitaires ou gouvernementaux mettent en ligne des formulaires ou documents officiels sous le format PDF. Les moteurs de recherche revoient ainsi leur stratégie. Google est celui qui prend en charge le plus de formats différents, parmi la vingtaine de formats indexés figurent ainsi les documents PDF, Postscript, mais aussi Word, Excel, Lotus, PowerPoint ou Works.

Les pages générées dynamiquement

Aux premiers temps du Web, les sites étaient constitués simplement d'une collection finie de pages HTML selon une hiérarchie arborescente. Pour un site de petite

taille qui ne nécessite pas de mises à jour très fréquentes, cette structure est la plus facile à gérer et elle est encore utilisée aujourd'hui par un très grand nombre de sites. Cependant, lorsque le site comprend des milliers de pages qui se référencent entre elles et qui doivent être remises à jour régulièrement, il devient quasiment impossible de maintenir la cohérence du site en utilisant de simples pages statiques. D'une part le site devient assez encombrant sur le serveur Web, d'autre part la modification d'une page, par exemple sa suppression, doit être répercutée sur d'autres pages afin de conserver des liens valides. Depuis quelques années, il est possible de créer des sites Web dynamiques. Contrairement aux sites statiques, les sites dynamiques ne maintiennent pas de version de leurs pages sur le serveur, elles ne sont générées qu'au moment où l'on veut y accéder. Lorsque l'on veut consulter une page, le serveur exécute un script (PHP, CGI, ASP, JSP) qui définit la forme et le contenu de la page. Le serveur conserve séparément les modèles de pages Web et les données qui sont souvent stockées dans des bases de données. Cette technologie permet de programmer les pages Web et donc de les personnaliser ou de maintenir la cohérence du site facilement. Elle permet également de disposer de la puissance et de la souplesse des bases de données afin de gérer efficacement de grandes quantités de données.

On peut distinguer trois principaux types de sites dynamiques. Certains sites utilisent des bases de données non pas pour la gestion de grandes quantités de données ou la possibilité d'avoir recours à des requêtes complexes, mais plutôt car cette structure est plus robuste et permet de maintenir la cohérence du site facilement. De plus les pages dynamiques permettent aux visiteurs du site d'interagir, en laissant des messages sur des forums ou bien sur des livres d'or. Des sites commerciaux, d'associations ou encore des sites personnels un peu élaborés entrent dans cette catégorie. Par exemple, le site personnel « Les Meilleurs Groupes Français du Monde! » (<http://joune.free.fr/>), possède de nombreuses fonctionnalités et la navigation y est très agréable.

Dans de tels sites, la navigation se fait principalement en cliquant de liens en liens, mais pourtant leur contenu n'est en général pas indexé par les moteurs de

recherche. Chaque lien contient une URL dynamique souvent reconnaissable par la présence du caractère « ? ». Cette URL lance l'exécution d'un script permettant de générer la page voulue. Cependant, un programmeur mal intentionné peut facilement créer un script qui va générer une infinité de pages Web afin d'inonder le crawler, ou bien l'entraîner dans une boucle infinie. Le crawler sera alors piégé et consommera inutilement beaucoup de ressources. De plus, le crawler ignore le résultat d'un script avant son exécution. Ainsi, les concepteurs de moteurs de recherche ont fait le choix de ne pas indexer les pages Web dont l'adresse est dynamique.

Un second type de sites fournit de l'information très souvent renouvelée voire en temps réel. Ce sont des sites qui proposent les cours de la bourse, des informations météorologiques, les horaires d'arrivées des avions, ou bien encore des résultats sportifs. Cette information, qui est en général de taille importante, change rapidement et perd de sa valeur avec le temps. Le délai de rafraîchissement des moteurs de recherche étant de l'ordre de quelques semaines, il est inutile d'indexer cette information par les moteurs, car elle serait complètement caduque.

Le troisième type de sites utilise pleinement les capacités de la technologie des bases de données. Ces sites gèrent de grandes quantités de données auxquelles on peut accéder par des requêtes complexes. Pour extraire de l'information, il faut interagir avec la base de données. L'interface est en général réalisée sous la forme d'une page HTML contenant un ou plusieurs champs de recherche et des fonctions de recherche plus ou moins élaborées. Le contenu de ces sites représente la partie du Web invisible la plus intéressante, de part sa taille et la qualité de l'information présente. Entrent dans cette catégorie, les encyclopédies en ligne, les sites de bibliothèques, les services de pages jaunes ou les catalogues de boutiques en ligne. Les sites Amazon.com ou eBay.com sont de très importantes sources de données. De plus, la plupart des institutions de recherche ont mis leur bases de données en ligne, et ceci, de manière publique, c'est-à-dire que l'on peut y accéder sans frais, ni inscription. On a ainsi à disposition sur le Web une source d'information spécialisée à la pointe de la recherche mondiale. Par exemple, le

site de la NASA (www.nasa.gov) met en ligne plusieurs bases de données et un grand nombre d'informations sur ses différents projets. L'Ifremer (Institut français de recherche pour l'exploitation de la mer) permet d'accéder via son site Web, à des bases de données océanographiques.

Les crawlers peuvent accéder à la page d'accueil d'un site dynamique, mais ils sont incapables de remplir les champs afin de générer les pages dynamiques qui ne peuvent donc pas être indexées. Le problème provient de la grande variété des champs et méthodes d'accès de ces sites. Il n'y a pas de modèle unique, chaque site possède sa propre interface d'accès. Des projets tentent de créer des crawlers plus intelligents capables de remplir les formulaires d'accès afin d'indexer le contenu des sites. Certains sont préprogrammés pour interagir avec certaines bases de données, mais dans ce cas, il faut déjà connaître les sites à indexer, et la procédure n'est pas entièrement automatisée. Une autre approche consiste à deviner les champs à remplir. Cependant, on n'aura pas une efficacité aussi bonne qu'avec les outils originaux de la base de données.

2.3.2 Classification du Web invisible

Chris Sherman et Gary Price ont établi une classification du Web invisible dans leur ouvrage « The Invisible Web » [12]. Ils le décomposent en quatre types : le Web opaque, le Web privé, le Web propriétaire, et le vrai Web invisible.

Le Web opaque

Il est constitué des pages qui, techniquement peuvent être parcourues par les crawlers et être indexées, mais qui ne le sont pas. Il existe plusieurs raisons :

- les pages sont trop profondes, mais comme les coûts de parcours et d'indexation diminuent, ce problème tend à disparaître;
- les pages sont nouvelles ou ont été changées depuis le dernier passage des crawlers, elles sont invisibles jusqu'au prochain passage des crawlers;

- la majorité des moteurs de recherche limite le nombre de résultats retournés lors d'une requête. Ainsi pour une requête à laquelle correspondent 10 millions de documents indexés, le moteur ne retournera que mille réponses, les autres restant invisibles. Ce problème peut être contourné en effectuant une requête plus précise;
- Les pages sont isolées, mais on peut les soumettre manuellement.

Le Web opaque est donc assez large mais ne pose pas de problèmes majeurs.

Le Web privé

Le Web privé est constitué de pages Web qui sont techniquement indexables mais qui sont exclues de l'index des moteurs de recherche directement par les concepteurs des sites. On peut, soit insérer la balise *noindex* dans l'en-tête de chaque page qui indique aux crawlers de ne pas lire la suite, soit utiliser le fichier *robots.txt* que l'on place à la racine du site. On y indique les parties du site que l'on ne veut pas voir indexées.

Le Web privé ne constitue pas un sujet de recherche, puisque ces pages ne sont pas indexées délibérément.

Le Web propriétaire

Les sites du Web propriétaire nécessitent un enregistrement préalable afin d'accéder à leur contenu. Dans la plupart des cas, l'enregistrement est gratuit mais les crawlers sont techniquement incapables de réaliser ce processus. Par exemple, le site du journal « Le Monde » possède une partie publique et une partie réservée aux visiteurs enregistrés offrant des rubriques supplémentaires.

Le vrai Web invisible

C'est la partie du Web qui n'est pas indexable pour des raisons purement techniques. Nous avons vu que de nombreux formats étaient difficilement ou pas du tout

indexables. Les pages Web dynamiques sont ignorées pour des raisons de sécurité. De plus, l'information contenue réellement dans les bases de données n'est pas accessible par les moteurs de recherche. C'est dans ce domaine que la technologie de la recherche d'information a à progresser.

2.3.3 Les outils de recherche sur le Web invisible

Il existe deux types d'outils conçus afin d'effectuer des recherches sur le Web invisible. Nous parlons ici d'outils de recherche sur les sites dynamiques basés sur des bases de données, car ils constituent la part la plus intéressante du Web invisible.

D'une part, il y a des sites portails qui répertorient manuellement un très grand nombre de sites du Web invisible. Les sites sont classés en catégories et sous-catégories, et il est possible d'effectuer des recherches par mots-clés à l'intérieur du portail. En fait, il s'agit simplement d'annuaires comme Yahoo! Mais qui sont spécialisés dans le Web invisible. Nous pouvons citer les sites « InvisibleWeb.com » et « Invisible-Web.net » qui regroupent quelques milliers de sites du Web invisible.

D'autre part, des méta-moteurs sont spécialisés dans l'exploration du Web invisible. En plus d'effectuer des recherches sur des moteurs de recherche traditionnels, ils sont capables de lancer des requêtes sur des sources d'information du Web invisible. Par exemple, « ProFusion » (www.profusion.com) accède à plus de mille bases de données du Web invisible. On peut affiner sa recherche en sélectionnant les catégories pertinentes, et on peut aussi choisir l'ensemble des sites sur lesquels on lance la recherche simultanée.

Ces outils facilitent énormément la recherche d'information sur le Web invisible. Un utilisateur averti peut ainsi trouver de l'information spécialisée très rapidement. Cependant, il est nécessaire de bien formuler sa recherche, c'est-à-dire de choisir les catégories de recherche les plus pertinentes, si l'on veut voir sa requête aboutir. De plus, ces deux outils fonctionnent grâce à un référencement manuel des sites. Ils sont donc limités par ce facteur humain. Mais il est à noter que les sites auxquels on s'intéresse

sont en général volumineux et reconnus dans leur domaine. Ils sont donc en nombre beaucoup plus limité que des sites personnels et sont ainsi plus facilement référençables manuellement.

2.3.4 Les évolutions futures du Web invisible

Le Web invisible tend à la fois à rétrécir et à grandir. D'une part, le coût des ressources diminuant et de nouvelles technologies apparaissant, on sera plus à même de révéler des parties aujourd'hui cachées du Web invisible. Mais d'autre part, le Web invisible ayant un taux de croissance très élevé, il paraît difficile de parer cette croissance. Nous allons répertorier les différentes techniques qui semblent prometteuses pour la recherche d'information sur le Web invisible.

Amélioration des crawlers

Les coûts de calcul, de stockage et de bande passante diminuant, les moteurs de recherche vont moins limiter le nombre de pages à indexer. Mais parallèlement, la taille du Web augmentant, le délai de rafraîchissement des pages ne peut pas être réduit suffisamment. Les crawlers doivent donc adapter leur fréquence de passage suivant le type de sites parcourus. D'autres études essaient de se concentrer sur une portion de Web significative, sur laquelle on peut effectuer des recherches plus puissantes.

Usage des méta-données

Les méta-données sont un des champs de recherche les plus prometteurs. Elles permettraient de définir un standard sur la structure des données. La majorité des données présentes sur le Web aujourd'hui sont au format HTML qui structure uniquement la présentation du document et non son contenu. On a tenté d'ajouter des balises « Méta » aux pages HTML notamment pour les mots-clés et la description de la page. Mais les « spammers » en ayant abusé, les moteurs de recherche les ignorent souvent.

Le standard de méta-données semblant émerger est le *RDF* (Resource Description Framework) utilisant la syntaxe du *XML* (eXtensible Markup Language). Les méta-données permettraient de structurer l'information présente sur le Web et de la rendre compréhensible par les ordinateurs. On pourrait alors développer des outils beaucoup plus puissants. Cependant, les méta-données laissent aussi de nombreuses possibilités aux « spammers ». Il paraît alors nécessaire de mettre en place une autorité de certification des documents publiés sur le Web. Mais cette éventualité paraît peu probable.

Accès aux bases de données

Nous avons vu précédemment que de nombreuses pages dynamiques étaient indexables, mais que les moteurs choisissaient délibérément de les ignorer pour des raisons de sécurité. On pourrait adopter un code d'éthique et punir les sites piégeant les crawlers. Une solution plus réaliste a été mise en place par Inktomi qui n'indexe des sites dynamiques que contre une soumission payante. D'autre part, on peut considérer certains domaines du Web comme plus sûrs, et indexer leur contenu dynamique, comme par exemple les domaines .edu ou .gov.

Un point important est l'accès aux bases de données via un formulaire. Des agents capables d'interagir avec les formulaires de requêtes des bases de données, pourraient alors générer des pages Web indexables par le moteur de recherche. Il faut cependant noter que l'accès par l'intermédiaire d'un moteur de recherche sera moins puissant qu'un accès direct utilisant l'interface de la base de données.

Les langages de requête hypertexte

Des langages de requête se basant sur les liens hypertexte pourraient considérer le Web comme une seule base de données géante.

Le crawling en temps réel

À la place de parcourir le Web d'avance et de stocker l'information dans un index, on pourrait imaginer de ne parcourir le Web qu'au moment de la requête afin d'avoir l'information la plus fraîche possible. Ceci paraît irréalisable, mais un projet plus modeste d'IBM² considère les résultats d'un moteur de recherche et les vérifie en temps réel afin d'éliminer les liens morts ou incohérents, et de trouver plus de résultats.

On peut aussi faire de la recherche en temps réel, en lançant son propre agent de crawling sur un ensemble prédéfini du Web.

Formats multimédia

Il reste encore beaucoup à faire sur la recherche d'information sur support non textuel, notamment dans la recherche d'image.

2.4 Les agents mobiles et la recherche d'information

Le paradigme d'agent mobile est assez prometteur dans de nombreux domaines de l'informatique, bien qu'il n'ait pas encore fait ses preuves de manière convaincante. Nous allons tâcher dans cette section de définir le concept d'agent mobile, de discuter de ses avantages et de ses limites, et enfin nous le placerons dans le contexte plus spécifique de la recherche d'information.

2.4.1 Le concept d'agent mobile

Le concept d'*agent* est employé dans de nombreux domaines de l'informatique et des télécommunications et il recouvre des entités parfois assez différentes. Ce terme étant assez flou, il convient d'en adopter une définition précise. Une des définitions

² www.ibm.com/java/mapuccino/fetuccino

données par le « grand dictionnaire terminologique » de l'Office de la langue française, est la suivante :

« Entité physique ou virtuelle possédant des ressources propres, capable de percevoir son environnement, d'agir sur lui, de communiquer directement avec d'autres agents et dont les comportements visent à satisfaire ses propres objectifs. »

Dans notre cadre plus précis, nous adapterons cette définition comme le fait Greenberg [14]. Un agent est alors défini comme une entité logicielle qui :

- agit en faveur d'une autre entité (une personne ou bien un autre agent);
- est autonome et possède ses propres objectifs;
- interagit avec son environnement.

La *mobilité* n'est pas propre aux *agents mobiles*. Toute entité logicielle est potentiellement mobile et peut être transportée via différents supports (Disque dur, CD-ROM, réseau, etc...), entre deux lieux d'exécution. Il convient de distinguer deux types de mobilité :

- la *mobilité faible* selon laquelle un programme est envoyé d'un site à un autre, mais il reste sur le même site pendant toute la durée de son exécution. Seul le code et les données d'initialisation sont transférés;
- La *mobilité forte* qui concerne un programme, en l'occurrence un agent mobile, qui est capable d'entamer son exécution sur un site, de migrer sur un autre, et de reprendre son exécution exactement là où il l'avait interrompue. Dans ce cas, à la fois, le code, les données et l'état d'exécution sont transférés.

La plupart des entités logicielles ne disposent que de la mobilité faible. La mobilité forte est une migration beaucoup plus complexe. Les agents mobiles ont hérité des techniques de migration des *processus* entre deux ordinateurs. Un processus est une abstraction du système d'exploitation contenant le code, les données, et l'état d'exécution du programme. Cependant, cette migration requiert un système d'exploitation dédié à cause du transfert de ressources systèmes. Les agents mobiles ont

l'avantage d'être écrits dans des langages interprétés indépendants des machines. Ils peuvent être ainsi exécutés dans des environnements hétérogènes.

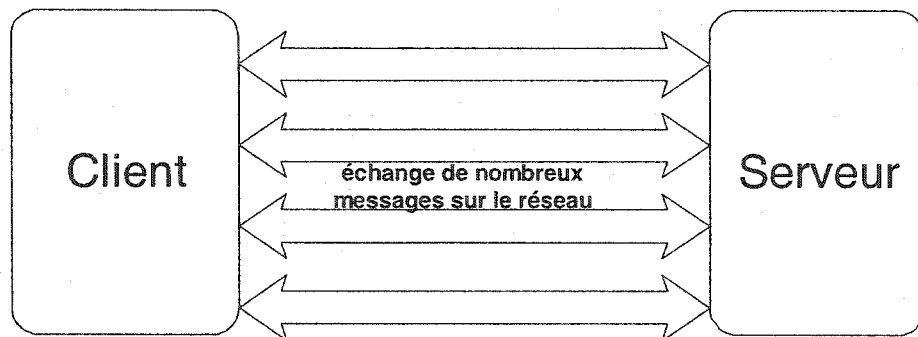
2.4.2 Les avantages et limitations des agents mobiles

La nature des agents mobiles, très différente du modèle client/serveur dominant les interactions sur les réseaux actuels, offre divers avantages. Cette technologie semble prometteuse. Cependant, elle tarde à s'imposer à cause de certaines limitations que nous allons développer par la suite.

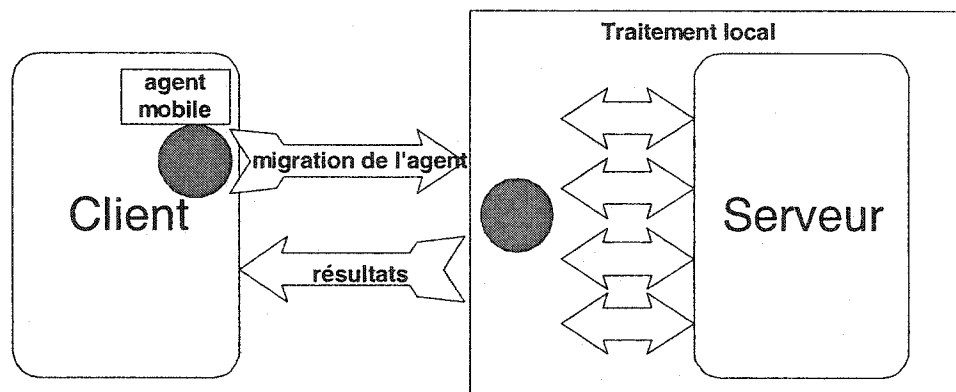
Avantages

Les agents mobiles ont potentiellement plusieurs avantages intéressants :

- *Représentation naturelle.* Un agent mobile étant une entité autonome agissant pour le compte d'une personne ou d'un autre agent, il peut être vu comme la représentation naturelle d'un individu et du monde réel. La programmation des agents mobile est donc assez intuitive.
- *Réduction de la charge réseau.* Un agent migre de nœud en nœud sur un réseau et effectue ses tâches localement. Alors que dans l'approche *client/serveur* de nombreux messages sont échangés sur le réseau, ici seul l'agent mobile est transféré (Figure 2.1). En fonction du rapport de tailles entre l'agent mobile et les données à traiter, l'approche agent mobile peut permettre une réduction significative de la charge réseau.
- *Personnalisation des services.* Dans l'approche client/serveur, un serveur peut mettre à disposition de l'utilisateur diverses fonctionnalités grâce à des *RPCs* (Remote Procedure Call). La programmation a lieu au niveau du serveur. Dans l'approche par agents mobiles, l'utilisateur programme lui-même son agent, qui exploite les *APIs* (Application Programming Interfaces) potentiellement mis à disposition par le serveur. L'utilisateur peut ainsi personnaliser à l'infini les traitements à réaliser sur le serveur.



Approche Client/Serveur



Approche Agents mobiles

Figure 2.1 Trafic généré par les deux approches

- *Connexion intermittente.* L'agent mobile étant autonome, l'utilisateur peut être déconnecté après avoir lancé son agent sur le réseau. Il suffit qu'il se reconnecte pour recueillir les résultats. Ceci est particulièrement utile lorsque l'on opère à partir de terminaux mobiles dont la connexion est peu fiable.
- *Survivabilité.* Ne dépendant pas d'un nœud particulier du réseau, l'agent mobile peut s'adapter si un nœud est défaillant.
- *Portabilité.* Les agents mobiles peuvent s'exécuter sur des plates-formes qui ne dépendent pas du système d'exploitation de la machine.

Lorsque l'on étudie les différents avantages des agents mobiles, on entrevoit des domaines d'application naturels où l'on pourrait les exploiter au mieux. Ainsi, l'utilisation des agents mobiles dans la recherche d'information semble prometteuse. En effet, si l'on peut envoyer un agent mobile sur un serveur pour analyser un grand nombre de données et pour retourner une réponse de petite taille, on aura un gain important en terme de communications sur le réseau. Un bon exemple est la recherche d'image sur des bases de données qui peut être intéressante dans un domaine comme l'imagerie médicale.

Les agents mobiles sont aussi un outil prometteur pour les terminaux mobiles ne disposant pas de grandes capacités de traitement, ni d'une connexion très fiable. Les agents mobiles permettraient de délocaliser les tâches à effectuer sur des serveurs plus conséquents.

Limitations

Malgré les avantages énumérés précédemment, cette technologie tarde à être exploitée pour les raisons suivantes :

- *Sécurité.* La sécurité est un problème majeur pour le développement de systèmes d'agents mobiles [15]. La mobilité du code pose surtout deux problèmes distincts. Tout d'abord la sécurité des plates-formes vis-à-vis d'agents malicieux est un problème est assez connu car il est similaire à la sécurité face aux virus informatiques ou bien face aux *applets Java*. Mais le problème le plus difficile est de protéger un agent contre une plate-forme malicieuse. En effet, celle-ci dispose d'un contrôle total sur l'environnement d'exécution de l'agent mobile. La plate-forme peut changer l'exécution de l'agent, retourner de faux résultats, accéder à des données confidentielles transportées par l'agent, tout simplement le détruire ou bien lui refuser des services. On comprend alors l'importance de ce point.

- *Manque d'interopérabilité.* Les agents mobiles ont besoin d'une *plate-forme* pour pouvoir s'exécuter. De nombreuses plates-formes sont développées [16](Aglet, Agent Tcl, Concordia, Tacoma, Voyager, Grasshopper, etc...), mais elles n'offrent pas toutes les mêmes interfaces, et les agents ne sont pas toujours programmés dans le même langage, bien que plusieurs utilisent le Java. Des initiatives de standard ont vu le jour. D'une part *FIPA* (Foundation for Intelligent Physical Agent) et d'autre part *MASIF* (Mobile Agent System Interoperability Facilities). Mais ces standards sont encore très peu appliqués.
- *Manque d'applications.* Il n'existe pas encore d'application utilisant les agents mobiles. Il n'y a pas, pour l'instant, d'application où l'usage des agents mobiles serait décisif (*killer application*).

Ces problèmes sont fortement liés entre eux. En effet, sans sécurité ni plates-formes compatibles, il est difficile de mettre en œuvre des applications en dehors des laboratoires. De même, la mise au point d'applications porteuses pourrait accélérer la standardisation des plates-formes et la recherche en matière de sécurité.

2.4.3 Les agents mobiles dans la recherche d'information

Les techniques de recherche d'information utilisent des architectures client/serveur mais aussi des agents intelligents. Par exemple, les crawlers des moteurs de recherche parcourent le Web selon une approche classique client/serveur, mais en fait ce sont des agents (statiques certes) autonomes qui collaborent entre eux. Un moteur de recherche comme Lycos [10] a une architecture multi-agents en ce qui concerne la collecte d'information. Ceci donne une grande souplesse à un système qui doit s'adapter à la grande taille et à l'hétérogénéité du Web.

L'utilisation des agents mobiles dans la recherche d'information permet de traiter l'information à la source. On peut exercer une recherche plus personnalisée, car on n'est pas limité par les fonctionnalités de recherche du serveur. De plus, les agents mobiles peuvent communiquer et collaborer entre eux ou avec des agents statiques. La recherche

peut se faire de manière asynchrone, et dépend peu de la qualité de la connexion. On peut aussi imaginer un agent mobile intelligent qui adapte sa recherche en fonction du profil de l'utilisateur et de son historique. Pour toutes ses raisons, le domaine de la recherche d'information semble le plus porteur pour l'utilisation des agents mobiles [17].

Plusieurs applications ayant recours à des agents mobiles ont été développées récemment. Certaines sont spécialisées dans des domaines particuliers, comme la recherche d'image médicale ou la recherche de rapports techniques [18]. D'autres, comme de la recherche d'information sur les pages XML, sont plus générales.

CHAPITRE III

RECHERCHE DYNAMIQUE SUR LE WEB INVISIBLE

Nous avons exploré les différents moyens d'effectuer une recherche sur des sites dynamiques à l'heure actuelle. Il existe de bons annuaires du Web invisible qui répertorient un très grand nombre de sites généralistes ou spécialisés. Cependant, nous atteignons toujours les limites des annuaires. Ils classent des sites Web par catégories, mais ils ne peuvent pas répondre à une recherche précise sur le contenu du site. Les méta-moteurs du Web invisible, quant à eux se limitent à des sites préétablis. Essayer de faire indexer par des moteurs de recherche le contenu des sites dynamiques nous a paru irréaliste compte tenu de la taille démesurée du Web invisible.

Aujourd'hui, il est tout à fait possible de rechercher de l'information sur le Web invisible. Mais seuls des utilisateurs avertis pourront le faire correctement en combinant leur connaissance du Web et les divers outils dont ils disposent. Par contre, il est difficile d'accès pour le grand public qui ne connaît pas forcément la nature du Web invisible et les moyens d'y accéder. Dans ce chapitre, nous proposons un modèle de recherche d'information sur le Web invisible, possédant la puissance et la simplicité d'utilisation des moteurs de recherche. Nous cherchons à rendre transparente, la recherche d'information sur le Web invisible, lequel deviendrait du même coup, visible. Nous avons alors pensé à effectuer une recherche dynamique sur des sites dynamiques comme le font les méta-moteurs, mais en adaptant cette recherche en fonction de la requête. Ceci est possible en exploitant les données de l'index d'un moteur de recherche. Notre outil pourrait s'intégrer aux moteurs de recherche Web.

3.1 Architecture générale

L'idée générale de notre architecture est d'ajouter à une architecture de base d'un moteur de recherche, un module qui va effectuer une recherche dynamique complémentaire à la recherche classique du moteur. Nous avons choisi d'utiliser les moteurs de recherche qui sont actuellement une technologie bien développée et très puissante.

La première idée fondamentale consiste à faire indexer par le moteur de recherche, les liens vers des sites dynamiques. Lors du parcours du Web (crawling) afin d'indexer son contenu, les crawlers vont de liens en liens, mais sont incapables d'accéder au contenu des sites dynamiques car il faut faire une requête afin de générer des pages correspondantes. Par contre, ils sont capables de repérer les pages formulaires qui sont possiblement des interfaces avec des bases de données. Sur les pages parcourues, nous indexons les liens hypertextes pointant vers des sites dynamiques, en plus du contenu de la page. Ainsi, lors d'une requête au moteur, il peut fournir une liste de liens vers des sites dynamiques, en fonction des mots-clés de la requête. Ceci constitue la première partie de notre architecture.

La deuxième partie vise à exploiter ces liens dynamiquement. Il s'agit d'analyser les pages vers lesquelles ils pointent. Il faut ensuite générer des requêtes en fonction de la page et des mots-clés. Puis, nous devons analyser le contenu des pages générées afin de mesurer leur pertinence par rapport à la requête. Finalement, il reste à compiler l'ensemble des résultats.

À partir de cette architecture générale, plusieurs variantes et scénarios sont possibles. Il est possible d'effectuer la recherche dynamique à partir d'un serveur dédié qui peut être ou non situé géographiquement au même endroit que le moteur de recherche. Mais il est également possible de lancer la recherche dynamique directement depuis la machine de l'utilisateur. Ces deux scénarios se déclinent sous des versions basées sur une architecture client/serveur et sous d'autres utilisant des agents mobiles.

3.2 Recherche à partir d'un serveur dédié

Dans ce modèle (figure 3.1), on ajoute au moteur de recherche, un module chargé de la recherche dynamique. Dans cette méthode, l'utilisateur continue de faire ses recherches par l'intermédiaire de l'interface Web du moteur de recherche. Tout est entièrement transparent pour lui, et il n'a besoin d'aucune configuration particulière de son terminal.

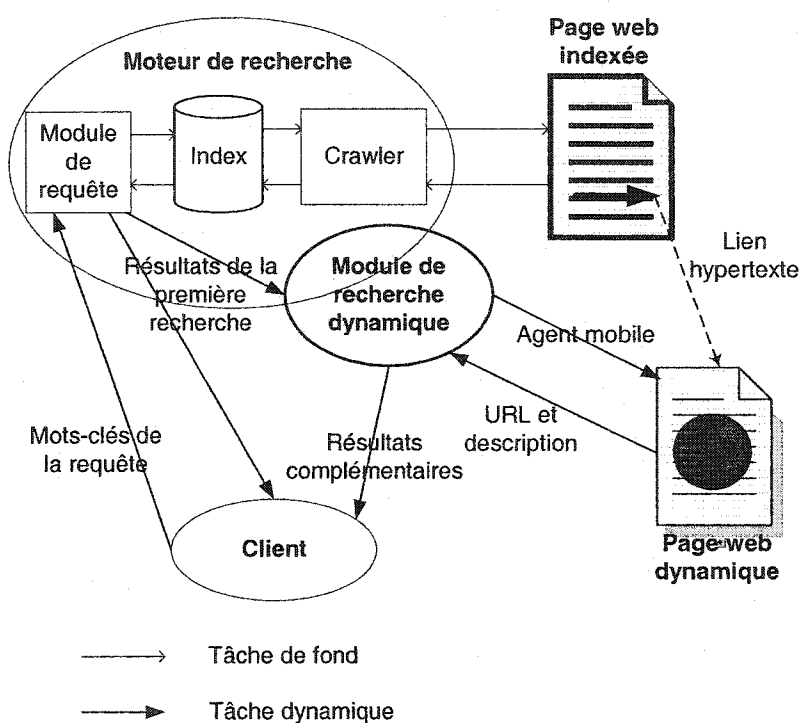


Figure 3.1 Architecture avec recherche dynamique à partir d'un serveur dédié

Grâce au crawling et à l'indexation effectués en tâches de fond, le moteur de recherche est capable de fournir en fonction de chaque requête, une liste de sites dynamiques susceptibles de fournir des résultats intéressants.

Le module de recherche dynamique prend alors en entrée les mots-clés de la requête et la liste en question. Il est chargé d'explorer dynamiquement les différents sites

et de fournir les résultats à l'utilisateur du moteur de recherche. Cette recherche dynamique peut se faire en utilisant le paradigme classique client /serveur, ou bien en ayant recours à des agents mobiles.

3.2.1 Approche client/serveur

Dans l'architecture client/serveur, le module de recherche dynamique communique avec les différents sites Web selon le paradigme traditionnel client/serveur. Il s'agit de la méthode classique d'accès aux sites Web.

Dans un premier temps, le module télécharge la page d'accès au site Web dynamique. Il doit ensuite analyser le contenu de la page téléchargée et surtout les formulaires qui y sont présents afin de pouvoir générer automatiquement une requête. Cette requête va avoir pour conséquence de créer une nouvelle page au niveau du serveur Web. Le module télécharge alors cette page et doit analyser son contenu afin de juger de sa pertinence.

Le module de recherche dynamique lance cette recherche sur tous les sites de la liste fournie par le moteur de recherche. Toutefois, si la liste est trop longue, on pourrait se restreindre à un sous-ensemble selon des critères à déterminer. Une fois tous les sites explorés, le module compile tous les résultats en une seule page Web à la manière des moteurs de recherche classiques.

3.2.2 Approche par agents mobiles

Dans ce modèle, les étapes de la recherche dynamique sont exactement les mêmes que dans le modèle client/serveur. La seule différence est que l'on ne communique plus avec les serveurs Web selon le paradigme client/serveur. Nous utilisons des agents mobiles qui vont traiter l'information localement, directement sur les serveurs Web.

Tout d'abord, le module de recherche dynamique envoie un agent mobile sur chaque site. Les agents sont alors autonomes. Ils analysent les pages Web localement, génèrent les pages correspondantes aux mots-clés de la recherche. Ils se contentent de

renvoyer au module, l'URL de la page générée, ainsi qu'une courte description de la page. Le module peut alors, comme précédemment, compiler les résultats sous la forme d'une page Web afin de l'envoyer à l'utilisateur.

3.2.3 Avantages et inconvénients des deux approches

Les deux approches expliquées précédemment ont chacune leurs intérêts et leurs désavantages. En effet, l'approche client/serveur a fait ses preuves, et est aujourd'hui utilisée par la plupart des applications communicant sur un réseau. Il existe des protocoles standardisés comme le protocole http pour le Web, et aussi de nombreuses plates-formes bien développées, comme Mozilla côté client ou Apache côté serveur. L'aspect sécurité est assez bien développé, bien que toujours imparfait, aussi bien pour les nœuds communicants que pour le canal de communication. Cette approche nous permet donc de nous baser sur une technologie stable et bien établie. Par contre, dans ce modèle, la communication client/serveur engendre beaucoup de trafic. On doit en effet analyser le contenu de plusieurs sites pour finalement produire une seule page de résultats. De plus, le module de recherche dynamique doit disposer d'une grande capacité de calcul afin d'explorer en parallèle différents sites par demande, tout en faisant face à de nombreuses demandes.

À l'inverse, les agents mobiles sont une technologie récente, qui n'a pas encore fait ses preuves dans des applications pratiques à grande échelle. Aucune plate-forme ne s'est encore réellement imposée. Actuellement, tous les sites Web sont implémentés suivant l'approche client/serveur, et aucun site courant ne supporte les agents mobiles. La sécurité est un élément crucial qui n'est pas encore traité de façon satisfaisante. Par contre, comme nous l'avons vu dans le Chapitre 2, les agents mobiles semblent particulièrement bien adaptés au domaine de la recherche d'information. Dans notre modèle, ils permettent un gain de trafic pouvant être significatif. En effet, on envoie l'agent mobile vers le site Web, puis seuls les résultats de la recherche sont retransmis vers le module de recherche dynamique. Si la taille de l'agent mobile est suffisamment

petite et que la quantité d'information à traiter localement est suffisamment grande, alors il y aura gain en trafic. De plus, comme le traitement de l'information est décentralisé sur les différents serveurs Web, on évitera une surcharge de calcul au module de recherche dynamique. Un autre avantage des agents mobiles, est qu'ils peuvent effectuer des opérations personnalisables à volonté sur le serveur, alors que dans l'approche client/serveur, on reste limité par les opérations préprogrammées au niveau du serveur.

3.3 Recherche à partir du terminal de l'utilisateur

Dans ce nouveau modèle (Figure 3.2), nous reprenons le modèle précédent, mais en délocalisant le module de recherche dynamique sur le terminal de l'utilisateur. Pour l'utilisateur, le service se présenterait sous la forme d'un logiciel à installer sur sa machine, à la manière des métamoteurs logiciels comme Copernic. Ce serait un logiciel dépendant du moteur de recherche offrant le service de recherche dynamique. Lors d'une recherche classique sur le moteur de recherche, on pourrait effectuer la recherche dynamique complémentaire en cliquant sur un lien qui lance le logiciel et télécharge la liste des sites fournie par le moteur. L'intérêt de cette méthode est de permettre la personnalisation et un meilleur contrôle de la recherche par l'utilisateur, car la recherche est lancée directement depuis sa machine, et non par l'intermédiaire d'un serveur dédié.

3.3.1 Approche client/serveur

La seule différence avec le modèle précédent est qu'il n'y a plus de module de recherche dynamique sur un serveur dédié, mais un logiciel de recherche dynamique directement installé sur la machine de l'utilisateur.

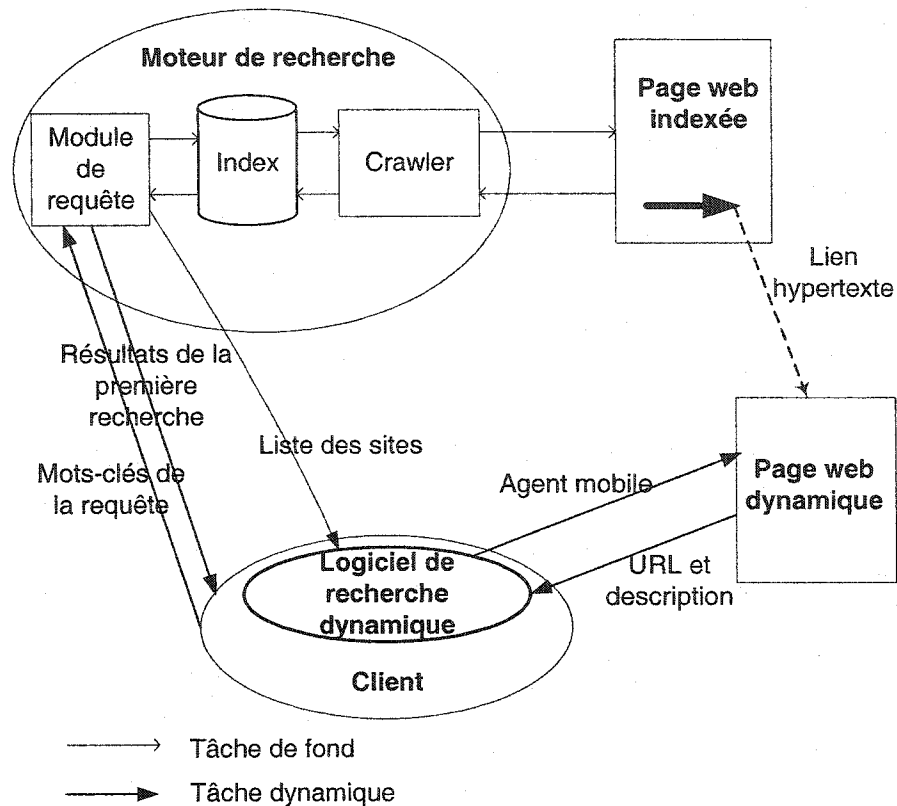


Figure 3.2 Architecture avec recherche dynamique à partir de la machine utilisateur

3.3.2 Approche par agents mobiles

Dans un premier temps, nous pouvons imaginer un modèle qui serait le pendant au modèle client/serveur précédent. Il suffit de remplacer les interactions sous la forme client/serveur entre le logiciel et chaque site Web, par l'envoi d'un agent mobile sur le site. Le terminal de l'utilisateur envoie alors un agent mobile pour chaque site à visiter.

Mais afin de profiter pleinement de la puissance des agents mobiles, il serait plus judicieux de n'envoyer qu'un agent qui se dupliquerait sur le réseau. En profitant du fait que la communication par agents mobiles est asynchrone, on donne alors accès à ce service à des utilisateurs possédant des terminaux à faible et/ou mauvaise liaison.

Le trafic sur le réseau suit alors le modèle suivant (Figure 3.3). L'agent mobile envoyé depuis le terminal utilisateur est dupliqué sur une plate-forme du réseau. Cette plate-forme offre aux agents mobiles des possibilités d'exécution. Ce genre de service est souvent présent dans les architectures ayant recours aux agents mobiles. Ensuite, chacun des clones migre vers un site Web, afin de s'y exécuter. Finalement, chaque agent envoie le résultat de sa recherche à l'utilisateur sous la forme d'une URL et de la description du site. On peut aussi imaginer que l'agent d'origine collecte les résultats pour les envoyer en une seule fois au terminal de l'utilisateur.

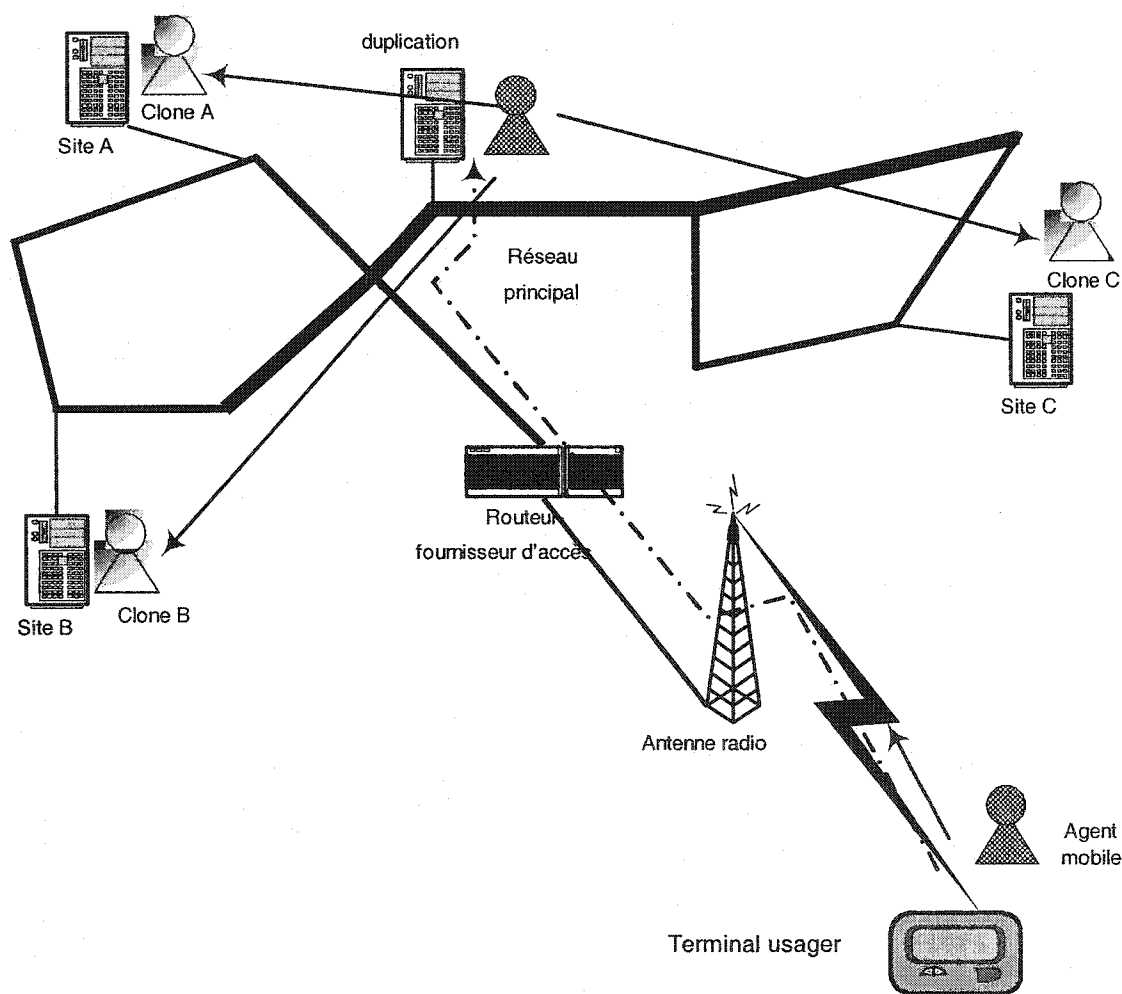


Figure 3.3 Parcours des agents mobiles

3.3.3 Avantages et inconvénients des deux approches

Les points soulevés lors de la recherche par serveur dédié sont toujours aussi pertinents. De plus, dans cette méthode où la recherche se fait à partir du terminal de l'utilisateur, les agents mobiles sont beaucoup mieux adaptés aux cas où l'utilisateur possède un terminal avec de faibles capacités de traitement ou une liaison de mauvaise qualité. Ainsi, un utilisateur possédant un PDA ayant une liaison sans-fil intermittente peut envoyer un agent mobile, se déconnecter quelques instants, et récupérer les résultats de la recherche. Et ceci, sans consommer beaucoup de ressources au niveau du terminal et de sa bande passante.

3.4 Intérêts et limites

Le but principal de la méthode élaborée est de développer un outil de recherche performant sur le Web invisible. Notre modèle s'intègre à un outil déjà existant et très performant : le moteur de recherche. L'originalité de la méthode est qu'elle s'appuie sur les données présentes dans l'index du moteur de recherche afin de choisir automatiquement les sites du Web invisible sur lesquels effectuer la recherche en fonction des mots-clés de la requête. Ceci les différencie des méta-moteurs de recherche du Web invisible, qui lancent leurs recherches sur une liste de sites fixe et limitée.

La recherche est effectuée de manière dynamique, c'est-à-dire qu'on ne lance la recherche sur les sites Web qu'au moment de la requête d'un usager. Le moteur de recherche n'est pas capable de parcourir les sites du Web invisible car il effectue le parcours du Web en tâche de fond, en permanence. Ces crawlers essaient de parcourir exhaustivement le Web sans but précis, afin de l'indexer. Alors qu'avec notre méthode, lors d'une recherche dynamique, nous connaissons, contrairement aux crawlers des moteurs, les mots-clés de la requête de l'utilisateur. C'est ce qui nous permet d'effectuer une requête nous donnant accès au contenu du site. De plus, comme la recherche est dynamique, on n'a pas besoin de stocker les données parcourues. De toutes façons, il

paraît irréaliste, compte tenu de sa taille, de vouloir stocker le contenu du Web invisible dans un index. Un autre avantage à cette recherche dynamique, est qu'elle fournit une information fraîche, à la différence des moteurs de recherche qui peuvent fournir des résultats correspondants à des pages indexées plusieurs semaines auparavant. Notre méthode garantit des liens valides dans son résultat. Nous pouvons ainsi effectuer des recherches sur des thèmes qui sont mis à jour plus fréquemment que l'index des moteurs de recherche. Nous pouvons lancer une recherche sur un sujet d'actualité, alors que les moteurs de recherche classiques ne donneront que des résultats caducs, ou des résultats qui ne périment pas.

Bien évidemment, on peut déceler quelques limites dans notre modèle. Tout d'abord, effectuer une recherche dynamiquement, ne comporte pas que des intérêts. Cette recherche entraîne un inévitable délai. Ce délai n'est pas nécessairement prohibitif, mais il faut qu'il reste dans des limites raisonnables, typiquement de l'ordre de la minute. Notre méthode développe une méthode *complémentaire*. L'utilisateur lance d'abord une recherche classique sur un moteur de recherche. Cette première recherche est très rapide, en général de l'ordre de la seconde. Ensuite, s'il veut lancer une recherche plus poussée, il est moins gêné qu'elle prenne plus de temps.

Une limite possible de notre méthode est qu'elle consomme plus de ressources au niveau des serveurs Web, que les moteurs de recherche traditionnels. À chaque requête, il y a des recherches effectuées sur des sites Web, contrairement aux moteurs de recherche qui parcourent leur propre index. Mais aussi, dans l'approche par agents mobiles, le traitement des données est décentralisé sur les serveurs des sites Web, ce qui engendre une consommation de leurs ressources. Ceci pourrait déplaire à des hébergeurs Web qui seraient tentés d'interdire l'accès des sites qu'ils hébergent aux agents mobiles de notre outil de recherche.

Une troisième limite de notre méthode est qu'elle met au point un outil de recherche complémentaire. Il faut donc que cette recherche complémentaire apporte,

dans certains cas, des résultats plus pertinents que la première recherche. Si la recherche préliminaire apporte des résultats satisfaisants, la recherche complémentaire n'a que peu d'intérêts. Par contre, lorsque la première recherche n'est pas satisfaisante, la recherche complémentaire devrait apporter des résultats plus pertinents. Cependant comme la liste des sites dynamiques à explorer provient du même moteur qui fournit les résultats de la première recherche, il peut y avoir le danger que l'on ne cherche pas dans une zone plus large. Ce facteur est assez difficile à évaluer de manière théorique.

3.6 Améliorations possibles

Nous avons développé une méthode de recherche sur le Web invisible, autour d'une idée assez simple. Ensuite nous avons proposé plusieurs architectures possibles, chacune présentant ses avantages, bien que les modèles basés sur une approche par agents mobiles semblent plus prometteurs. Nous suggérons ici quelques idées d'amélioration que toutefois, nous n'aborderons pas dans l'implémentation de notre méthode.

Une amélioration assez simple consisterait à instaurer un système de cache afin de répondre à des requêtes fréquentes. Le problème de l'utilisation d'une mémoire cache, est que l'information n'est pas forcément fraîche. Il y a donc toujours le problème de s'adapter à la mise à jour de l'information. Une façon astucieuse d'implanter cette fonction de cache serait d'incorporer à l'index du moteur de recherche les résultats provenant des requêtes très fréquentes. Ainsi, ces résultats seraient fournis à l'utilisateur dès la première recherche classique.

Un point intéressant serait de créer des agents mobiles évolutifs, c'est-à-dire qu'ils pourraient s'adapter aux sites dynamiques. Chaque site est programmé de façon différente, et il est intéressant d'avoir des agents mobiles personnalisés en fonction des sites à explorer. On pourrait alors constituer une bibliothèque d'agents mobiles adaptés à chaque site fréquemment visité. Ce point nécessite d'implémenter des notions d'intelligence artificielle et d'apprentissage.

Un troisième élément intéressant concerne la sémantique des mots et son utilisation dans la recherche d'information. Les premiers moteurs de recherche étaient basés sur des critères purement syntaxiques, mais afin de donner des résultats toujours plus pertinents, ils essaient de plus en plus de tenir compte d'éléments sémantiques. Nous avons parlé, dans le chapitre précédent, du nouvel outil de recherche Exalead qui permet d'affiner sa recherche selon des critères sémantiques. Ainsi, notre modèle pourrait bénéficier d'une liste de sites plus pertinents, si le moteur de recherche pouvait élargir cette liste en travaillant par rapprochement sémantique sur les mots-clés de la requête originale.

CHAPITRE IV

IMPLÉMENTATION ET MISE EN ŒUVRE

Ce chapitre vise à détailler l'implémentation de l'architecture théorique que nous avons développée dans le chapitre précédent. Le but de ce travail pratique est de pouvoir effectuer des expérimentations qui démontrent la pertinence de notre modèle, et qui permettent de comparer les performances relatives des approches client/serveur et agent mobile pouvant être employées dans notre outil de recherche d'information. Nous allons tout d'abord présenter l'environnement d'implémentation, puis nous expliquerons nos choix concernant la délimitation et les simplifications de la partie programmée, pour enfin analyser les détails de l'implémentation.

4.1 Environnement de réalisation

Nous allons expliquer brièvement nos choix quant à l'environnement de programmation, c'est à dire essentiellement le langage de programmation et la plate-forme d'agent mobile utilisés.

4.1.1 La plate-forme d'agent mobile Grasshopper

Nous avons choisi de travailler avec la plate-forme d'agent mobile Grasshopper (<http://www.grasshopper.de>), entièrement programmée en Java et qui est supportée sur des systèmes Unix, Windows, et Windows CE qui équipe des appareils portatifs ou embarqués. De plus, à l'heure actuelle, Grasshopper est la seule plate-forme respectant les standards MASIF (Mobile Agent System Inter-operability Facility) de l'OMG³ et

³ Object Management Group (<http://www.omg.org>)

FIPA⁴ (Foundation of Intelligent Physical Agents) visant à spécifier des normes permettant l'interopérabilité entre les différents systèmes d'agents.

4.1.2 Le langage de programmation Java

Les agents mobiles Grasshopper se programmant en Java, nous avons naturellement choisi de programmer l'ensemble de notre travail en utilisant ce langage. De plus, ce langage extrêmement portable est particulièrement adapté pour des applications Web s'exécutant sur des serveurs qui peuvent être installés sur des systèmes d'exploitations divers.

Les différentes classes développées lors de l'approche client/serveur ont pu être utilisées sans modification dans l'approche agent mobile, ce qui a permis un gain de temps de développement appréciable.

Nous avons utilisé le logiciel de développement Jbuilder⁵ qui facilite beaucoup la programmation en Java.

4.1.3 Le système d'exploitation

Compte tenu de la portabilité du Java et de la plate-forme Grasshopper, le choix du système d'exploitation a peu d'importance. Nous avons pu développer notre application aussi bien sous un système Windows qu'Unix.

4.2 Choix d'implémentation

Lors de l'implémentation de notre modèle, nous avons dû faire des choix pour plusieurs raisons. Tout d'abord, nous nous sommes limités à une partie de l'architecture générale. De plus, nous avons dû effectuer des simplifications et préciser des détails non

⁴ <http://fipa.org>

⁵ Jbuilder de Borland (<http://www.borland.com/jbuilder>)

présents dans le modèle théorique. Dans cette section, nous allons expliquer ces choix et justifier leur pertinence dans l'optique d'expérimenter correctement notre modèle de recherche.

4.2.1 Délimitation de la partie à implémenter

Nous n'implémentons que la partie purement dynamique (Figure 4.1) de notre architecture. Nous n'intervenons pas au niveau du moteur de recherche. Il est difficile pour nous de modifier le fonctionnement interne d'un moteur de recherche. D'une part, les codes sources sont rarement disponibles publiquement. D'autre part, la mise en œuvre d'un moteur de recherche nécessite des ressources considérables aussi bien au niveau matériel qu'au niveau du personnel chargé du développement.

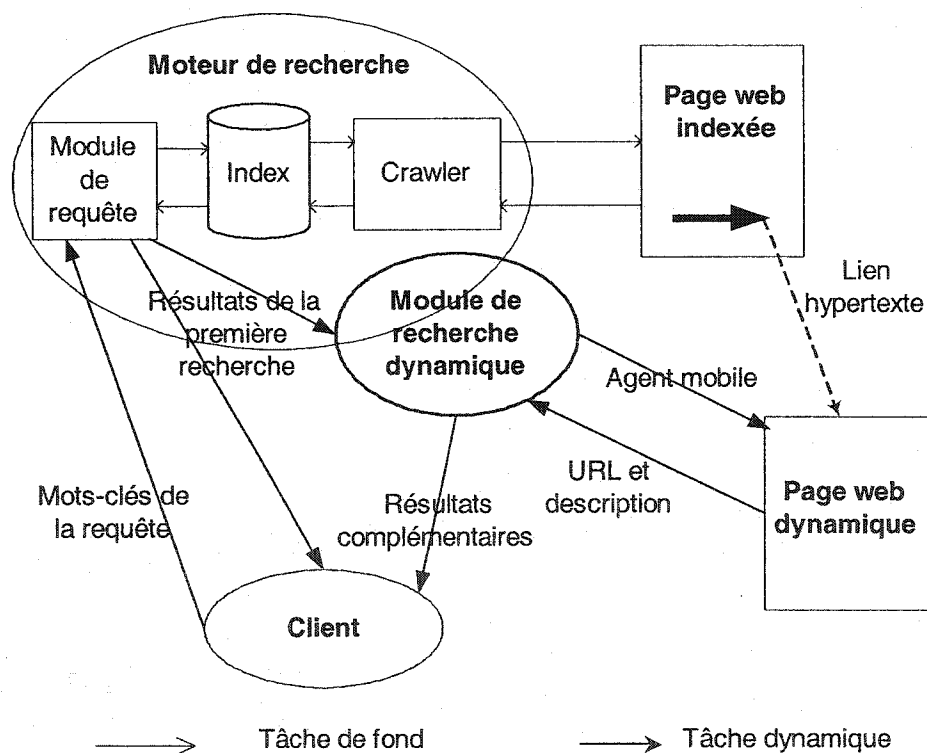


Figure 4.1 Sous-ensemble implémenté

Nous nous basons alors sur l'hypothèse que les concepteurs de moteurs de recherche sont capables de modifier leur indexage de manière à repérer et à stocker dans l'index, les liens vers des pages dynamiques, tout en les mettant en relation avec les mots-clés contenus dans les pages Web parcourues. Pour chaque requête, le moteur de recherche serait alors en mesure de fournir une liste de liens vers des pages dynamiques *en fonction* des mots-clés de la requête.

Le modèle implémenté (Figure 4.2) prend alors en arguments une liste des mots-clés représentant la requête et une liste de sites dynamiques supposée être fournie par un moteur de recherche. Ensuite, pour chaque site de la liste, on va analyser la page pointée qui doit contenir un formulaire. Il s'agit d'analyser ce formulaire afin de pouvoir générer une requête en fonction de nos mots-clés. On traite alors le contenu de la réponse pour juger de sa pertinence. Une fois la recherche effectuée sur chaque site, on compile tous les résultats et on les présente à l'utilisateur.

```
Recherche_dynamique(mots_clés, liste_sites){  
  Pour chaque site{  
    Analyse de la page-formulaire;  
    Requête(mots_clés);  
    Analyse de la page résultat;  
  }  
  Compilation des résultats;  
  Retourne page_résultats;  
}
```

Figure 4.2 Schéma général de l'application

4.2.2 Type de formulaire

Une des principales difficultés que nous avons rencontrées réside dans la diversité des formulaires présents sur le Web. Le langage HTML définit pourtant

clairement l'utilisation des formulaires grâce à l'utilisation de la balise *FORM* et de ses différents champs internes. Cependant, l'utilisation de *frames* peut compliquer considérablement l'analyse d'un formulaire, la page analysée étant alors constituée de plusieurs fichiers distincts.

Un autre problème plus complexe, car c'est un problème plus conceptuel que technique, consiste à identifier les différents champs d'un formulaire. En effet, un formulaire peut contenir divers champs correspondant, par exemple pour une bibliothèque, au titre ou à l'auteur d'un ouvrage. Chaque concepteur de site Web choisit les champs qu'il veut mettre et leur donne le nom qu'il désire. Il n'existe pas de standard quant à la dénomination ou l'utilisation des champs de recherche. L'identification des champs d'un formulaire est donc un problème complexe d'intelligence artificielle.

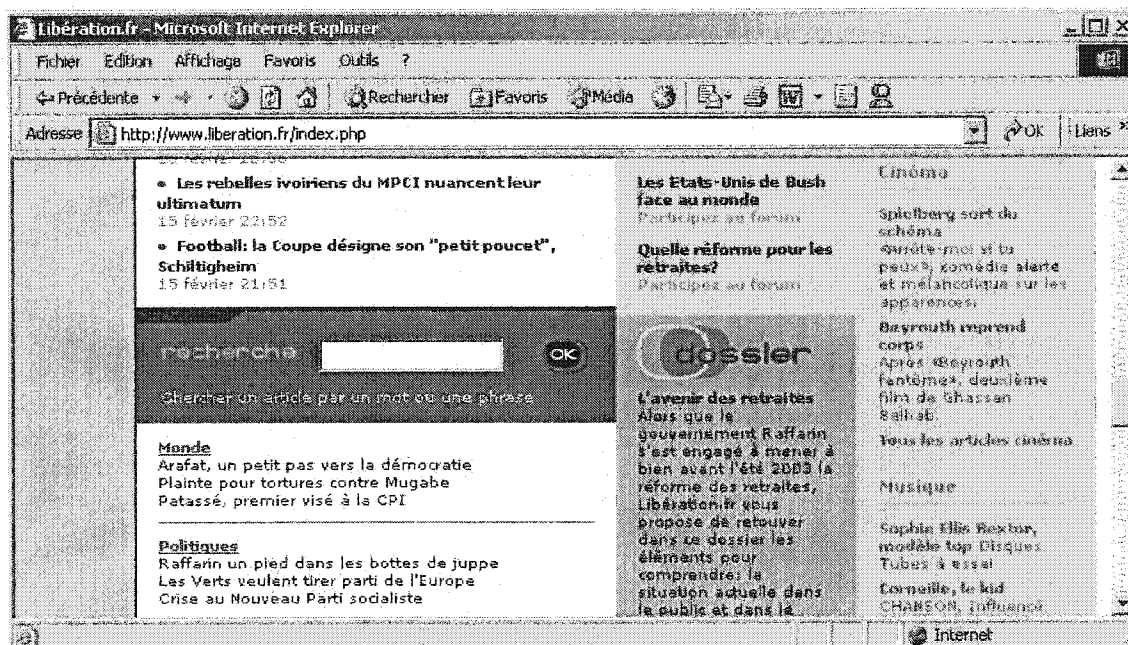


Figure 4.3 Formulaire sur la page d'accueil du site du quotidien Libération

Ainsi, face à ces problèmes, et en attendant l'hypothétique mise en place d'un standard normalisant les formulaires, nous avons choisi de nous limiter à l'analyse de formulaires simples. Les formulaires analysés sont présents sur des pages n'utilisant pas de frames, et ne comporte qu'un unique champ de recherche. Par contre, ils peuvent comporter de multiples champs concernant des choix de recherche, comme des listes défilantes ou des boutons radio. Les champs sont alors remplis en utilisant les valeurs par défaut.

Nous ne pouvons donc pas expérimenter notre modèle sur tous les types de sites dynamiques existants. Cependant, il nous permet d'analyser une large partie des sites présents sur le Web, comme par exemple le site du quotidien Libération (Figure 4.3).

4.2.3 Type d'expressions prises en compte

Les sites dynamiques permettent de faire des recherches avancées afin de répondre à des requêtes plus précises. La syntaxe correspondant aux types de recherche les plus courants est uniformisée. Pour des raisons de simplification technique, nous nous sommes limités à des recherches correspondant à un simple mot-clé ou à une expression comportant plusieurs mots.

4.2.4 Profondeur de recherche

En général, lorsque l'on effectue une recherche sur un site dynamique, on obtient d'abord une page comportant des liens vers les différents résultats. Et il est possible qu'il y ait encore sur ces pages d'autres liens vers des résultats plus approfondis. Une profondeur de recherche de deux ou de trois paraît alors nécessaire.

Pourtant, dans notre implémentation, nous avons choisi de nous limiter à une profondeur de recherche de un. Pour modifier notre modèle de façon à améliorer la profondeur de recherche, il suffit juste d'apporter des changements purement techniques à notre code. Cette complexification ne nous a pas paru nécessaire pour évaluer la pertinence de notre modèle, car il est possible de le tester sur des sites ne nécessitant

qu'une profondeur de recherche simple. D'autre part, du point de vue des performances en temps de calcul et en trafic généré, effectuer une recherche sur toute une branche d'un site équivaut à analyser une seule et même page de taille semblable à l'ensemble de la branche considérée.

4.2.5 Résultats de la recherche

Un des points clés en recherche d'information est la pertinence des résultats de la recherche. Dans le Chapitre II, nous avons évoqué ce point qui constitue maintenant un enjeu majeur pour les moteurs de recherche, et nous avons notamment présenté la technologie Exalead qui tente d'apporter de nouvelles solutions. Ce vaste champ de recherche dépassant le cadre de ce mémoire, nous avons choisi de juger de la pertinence des résultats seulement en vérifiant que les mots-clés de la requête de l'utilisateur étaient bien présents dans les pages analysées.

L'application que nous développons étant une application Web, nous présentons les résultats de la recherche sous la forme d'une page Web compilant les résultats issus des différents sites analysés. Pour chaque page pertinente, on indique le titre de la page ainsi que le lien correspondant ; afin de donner une description de la page à l'utilisateur, on y inscrit le texte entourant les mots-clés. Les mots-clés sont indiqués en gras. La Figure 4.4 présente une page compilant les résultats d'une recherche dynamique.

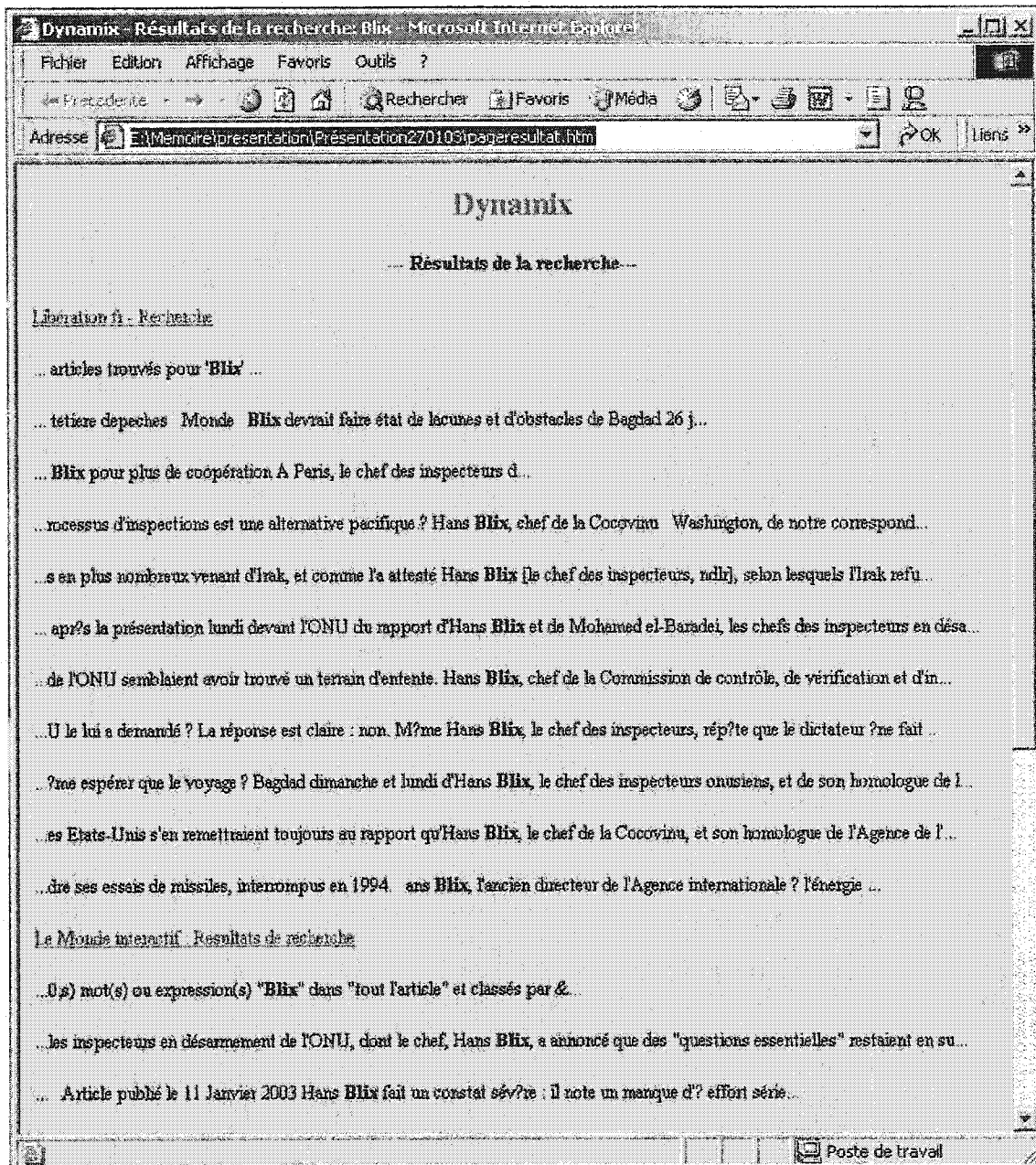


Figure 4.4 Page web compilant les résultats d'une recherche

4.3 Implémentation de l'approche client/serveur

La majeure partie de l'implémentation consiste à effectuer des requêtes au serveur Web et à analyser du code HTML. Il existe de nombreuses bibliothèques Java permettant d'effectuer ces tâches. Nous allons ainsi présenter d'abord la bibliothèque *HTMLParser* utilisée, avant de décrire plus en détail le « package » *Dynamix* que nous avons développé afin de réaliser notre implémentation.

4.3.1 La bibliothèque HTMLParser

HTMLParser est une bibliothèque Java « open source » développée selon la licence GNU et disponible sur le site de développement de programmes « open source » *SourceForge.net*⁶. Cette bibliothèque qui est encore actuellement en cours de développement permet de *parser*, c'est à dire d'analyser du code HTML. Nous avons utilisé la modification du 9 novembre 2002 de la version 1.2 de la bibliothèque.

Nous avons choisi d'utiliser cette bibliothèque car son code source est disponible, ce qui nous a permis de l'étudier correctement lorsque nous en avons utilisé différentes fonctions. De plus, elle permet d'analyser les balises HTML *FORM* depuis mai 2002. Et cette fonctionnalité marche vraiment correctement depuis quelques mois. La version du 9 novembre 2002 nous fournit les fonctionnalités suffisantes pour notre application.

La classe principale de la bibliothèque est la classe éponyme *HTMLParser* (Figure 4.5). Le constructeur de cette classe prend en argument l'URL de la page à analyser. Cette URL peut aussi bien être une adresse Web qu'un emplacement sur le disque dur de la machine locale. Un objet de cette classe peut appeler des méthodes permettant de parcourir la page HTML de façon structurée de nœud en nœud. Un nœud (classe *HTMLNode*) peut, soit être un élément de texte *HTMLStringNode*, soit une balise HTML *HTMLTag*.

⁶ <http://sourceforge.net/projects/htmlparser>

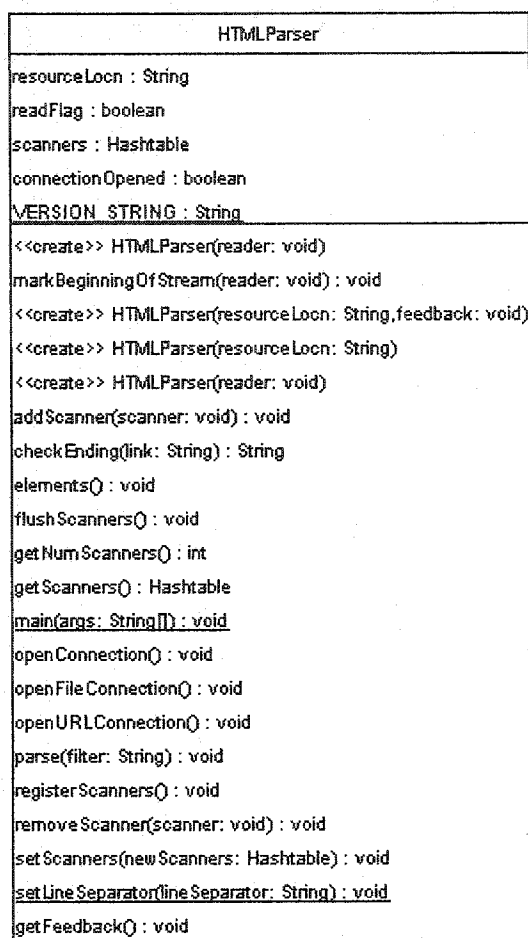


Figure 4.5 La classe HTMLParser

Pour chaque type de balise HTML, il existe une classe correspondante dans la bibliothèque *HTMLParser* qui tient compte de sa structure et de sa syntaxe. Cette classe contient des méthodes permettant d'extraire de la balise les données pertinentes.

Nous nous intéressons plus particulièrement à la classe *HTMLFormTag* (Figure 4.6) qui représente un formulaire HTML. Nous allons voir ci-dessous que les méthodes de cette classe nous permettent d'extraire du formulaire toutes les informations dont nous avons besoin afin de formuler notre requête.

HTMLFormTag
formURL : String formName : String formMethod : String formInputVector : Vector allNodesVector : Vector
<<create>> HTMLFormTag(formURL: String,formName: String,formMethod: String, formBegin: int,formEnd: int,tagLine: String,formInputVector: Vector,allNodesVector: Vector) getFormInputs() : Vector getFormLocation() : String getFormMethod() : String getInputTag() : void getFormName() : String setFormInputs(formInputVector: Vector) : void setFormLocation(formURL: String) : void setFormMethod(formMethod: String) : void setFormName(formName: String) : void toString() : String getAllNodesVector() : Vector setAllNodesVector(allNodesVector: Vector) : void toHTML() : String

Figure 4.6 La classe HTMLFormTag

FormRipper
mots : String
<<create>> FormRipper(resourceLocation: String,mots: String) rip() : Vector

StringExtractor
resource : String
mots : String
<<create>> StringExtractor(resource: String,mots: String) extractStrings() : String

PageGenerator
main(args: String[]) : void

Figure 4.7 Le package Dynamix

4.3.2 Le package *Dynamix*

Afin de réaliser l'implémentation de l'approche client/serveur de notre modèle, nous avons développé le package nommé *Dynamix* qui contient trois classes, et qui utilise la bibliothèque *HTMLParser*. Chaque classe effectue une fonction particulière.

La classe FormRipper

Le constructeur de cette classe prend comme arguments l'URL de la page à analyser et les mots-clés de la recherche. La méthode *rip()* permet d'analyser tous les formulaires contenus dans la page. Elle retourne un vecteur contenant, sous forme de *String*, les requêtes correspondant à chacun des formulaires.

Analysons la structure d'une balise HTML *FORM*. La Figure 4.8 présente le code HTML d'un formulaire simple inspiré de la page d'accueil du moteur de recherche Google.

```
<FORM name=f action=/search>
  <INPUT maxLength=256 size=40 name=q>
  <INPUT type=hidden value=UTF-8 name=ie>
  <INPUT type=hidden value=fr name=hl>
  <INPUT type=submit value=Recherche name=r>
  <INPUT type=submit value="J'ai de la chance" name=c>
  <FONT size=-1>
  Rechercher dans :
  <INPUT type=radio CHECKED value=Web name=meta>
  Web
  <INPUT type=radio value=lang_fr name=meta>
  Pages francophones
  </FONT>
</FORM>
```

Figure 4.8 Code HTML d'un formulaire

La balise formulaire comporte un champ *action* qui indique l'URL vers laquelle effectuer la requête. Un formulaire est constitué de plusieurs balises *INPUT* qui peuvent être de types différents. Suivant leur type, elles ont un rôle différent. Par exemple, une

balise INPUT de type *submit* représente un bouton à cliquer. Le code présenté correspond à la représentation graphique de la Figure 4.9.

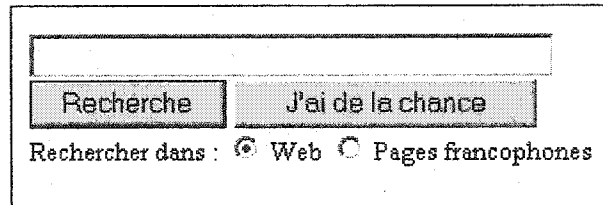


Figure 4.9 Un exemple de formulaire

Si l'on complète le champ de ce formulaire avec le mot-clé « mot_cle », la requête générée par le navigateur en appuyant sur le bouton « recherche », sera la suivante :

```
/search?hl=fr&ie=UTF-8&q=mot_cle&r=Recherche&meta=Web
```

La méthode *rip()* utilise les méthodes de la classe *HTMLFormTag* afin d'extraire ces informations des formulaires et de générer de telles requêtes.

La classe StringExtractor

La classe *StringExtractor* a pour but d'analyser une page susceptible de contenir des informations pertinentes par rapport aux mots-clés de la requête. Le constructeur de la classe prend donc en arguments l'URL de la page à analyser et les mots-clés de la requête.

La méthode *ExtractStrings* fait appel à la classe *HTMLParser* pour parcourir la page Web et l'analyser nœud à nœud. Comme on s'intéresse au contenu purement textuel de la page, on va se concentrer sur l'analyse des *HTMLStringNode* représentant les éléments de texte.

À chaque fois que l'on rencontre les mots-clés, on garde en mémoire une partie du texte les entourant. On concatène ces chaînes de caractères. Si l'analyse est fructueuse, on y ajoute le titre de la page et son URL. On retourne alors une unique chaîne de caractères formatée en HTML.

La classe PageGenerator

La classe *PageGenerator* contient la méthode principale *main* de notre programme. Ses deux arguments sont les mots-clés de type *String* et le nom d'un fichier *liste.txt* dont chaque ligne correspond à l'URL d'un site dynamique. Elle génère un fichier *resultat.htm* correspondant à la page Web compilant les résultats de la recherche dynamique (Figure 4.4).

Tout d'abord, la méthode *main* crée le fichier *resultat.htm* et génère l'en-tête HTML correspondant. Pour chacune des URL de la liste, elle appelle la méthode *rip()* de la classe *FormRipper*, puis pour chaque requête obtenue (qui correspond en fait à une URL), elle appelle la méthode *extractString* de la classe *StringExtractor*. Au fur et à mesure, on écrit les résultats de la méthode *extractString* dans le fichier *resultat.htm*. Si un site de la liste ne contient pas de formulaire, on indique « Le site visité ne permet pas d'effectuer de recherche ».

Pour lancer l'application, il faut inclure la bibliothèque *HTMLParser*, créer un fichier *liste.txt* avec la liste des sites à visiter, et enfin exécuter la classe *PageGenerator*.

4.4 Implémentation de l'approche agent mobile

L'implémentation de l'approche agent mobile ne nécessite pas une nouvelle implémentation depuis le début. En effet, nous nous sommes efforcés d'implémenter la version client/serveur de façon modulaire et en utilisant le langage Java, de manière à pouvoir réutiliser les classes que nous avons regroupées dans le package *Dynamix*.

Il s'agit donc d'intégrer les méthodes développées au sein de la structure d'un agent mobile Grasshopper. Deux possibilités se présentaient. La première consiste à

joindre le code des méthodes du package *Dynamix* à celui de l'agent mobile. Nous avons choisi la seconde possibilité, c'est à dire que nous avons considéré le package *Dynamix* comme une bibliothèque installée sur le serveur à laquelle nous faisons appel. Nous avons effectué ce choix par souci de simplification. Il ne nuit pas à notre modélisation car, dans la première possibilité, l'augmentation de la taille de l'agent mobile est fixe et ne change pas son ordre de grandeur.

4.4.1 Structure de l'agent mobile *DynamixMA*

Nous avons donc développé un agent mobile sous la forme d'une classe Java *DynamixMA* (Figure 4.10), héritée de la classe *agent.Mobile.Agent* de Grasshopper. Une telle classe est constituée essentiellement de la méthode *live()*, qui indique les différentes phases de vie de l'agent mobile :

- Phase 0 : lors de sa création sur la machine client, l'agent cherche à migrer sur le serveur du site à analyser.
- Phase 1 : l'agent s'exécute sur le serveur. Il appelle successivement les méthodes *rip* et *extractString* du package *Dynamix*. Il stocke les résultats de sa recherche dans la variable *reponse*. Il cherche enfin à retourner sur le client.
- Phase 2 : une fois retourné sur le client, l'agent crée une page Web *resultatMA.htm* compilant les résultats de la recherche, puis il met fin à son exécution.

```

import java.io.*;
import larim.kenzo.dynamix.*;    //package Dynamix
import de.ikv.grasshopper.communication.GrasshopperAddress;

public class DynamixMA extends de.ikv.grasshopper.agent.MobileAgent{

    int state=0;
    String URL="http://kenzo/sitetest/";    //Site dynamique à analyser
    String mots="agent";                    //mot-clé de la recherche
    String reponse;

    public void live() {    //méthode live
        switch(state){

            case 0:    //exécution sur le client : migration vers le serveur Web à analyser
                state=1;
                move(new GrasshopperAddress("socket://kenzo:7000/kenzo"));
                break;

            case 1:    //exécution sur le serveur : analyse à l'aide du package Dynamix
                state=2;
                FormRipper fr = new FormRipper(URL, mots);
                Vector v= fr.rip();
                StringExtractor se=new StringExtractor((String)v.elementAt(i), mots);
                reponse= se.extractStrings();
                move(getInfo().getHome());    //migration vers le client
                break;

            case 2:    //exécution sur le client : on écrit le résultat dans un fichier resultatMA.htm
                FileWriter page = new FileWriter("resultMA.htm");
                BufferedWriter output = new BufferedWriter(page);
                output.write("<html>... ");
                output.write(reponse);
                output.write("</body></html>");
                output.flush();
                output.close();
                remove();    //fin de l'exécution de l'agent
            }
        }
    }
}

```

Figure 4.10 Code simplifié de l'agent mobile DynamixMA

CHAPITRE V

EXPÉRIMENTATION ET RÉSULTATS

Ce chapitre est consacré à l'expérimentation visant à valider notre modèle théorique. Nous avons deux axes de tests. Tout d'abord, il s'agit de vérifier le fonctionnement de notre application dans l'environnement concret du Web. Ensuite, nous nous sommes surtout concentrés sur la comparaison des performances relatives des approches client/serveur et agent mobile. Nous allons donc présenter les tests qualitatifs sur des sites Web commerciaux, puis expliquer la configuration de l'environnement de tests, et enfin surtout détailler les plans d'expériences et les résultats de la comparaison des deux approches implémentées.

5.1 Tests qualitatifs

L'implémentation de notre modèle selon l'approche client/serveur a l'avantage de pouvoir être testée dans des conditions réelles. En effet, tous les sites Web opérationnels sont basés sur la technologie client/serveur. Nous avons donc lancé des recherches dynamiques simultanément sur plusieurs sites tels que Yahoo, le Quid, Libération ou le Monde. Nous avons obtenu les résultats escomptés dans des délais raisonnables pour une recherche en temps réel, c'est à dire entre 5 et 20 secondes.

Par exemple, en lançant une recherche simultanément sur les sites des quotidiens Libération et le Monde avec pour mot-clé *Blix*, nous obtenons dans un délai approximatif de 15 secondes, la page Web résultat présentée à la Figure 5.1. En le testant dans des conditions réelles, notre modèle nous donne donc satisfaction.

5.2 Configuration des machines tests

Afin de pouvoir réaliser des tests comparatifs entre nos deux approches implémentées, nous avons mis au point notre propre serveur supportant les agents mobiles. Nous décrivons ici les deux machines client et serveur utilisées. De manière assez réaliste, la machine utilisée en tant que serveur possède de meilleures ressources, et une connexion au débit supérieur ou égal à celle de la machine client. En plus de configurer correctement ces deux machines, nous avons développé un site Web test afin de l'installer sur le serveur.

5.2.1 Configuration du serveur

Le serveur a été installé sur un PC ordinaire et utilise des logiciels courants.

Ressources matérielles

Le serveur dispose d'un microprocesseur AMD cadencé à 1400 MHz, et de 256 Mo de mémoire vive. Il est relié à Internet au moyen d'une connexion ADSL domestique avec comme débits 1 Mbps pour la liaison descendante et 160 Kbps pour la liaison montante. De plus, il dispose d'une carte Fast Ethernet 100 Mbps permettant une connexion en réseau local.

Logiciels installés

Le serveur fonctionne avec le système d'exploitation Windows 2000 version professionnelle. Le serveur Web est composé d'un serveur Apache 2 couplé avec un serveur PHP 3. Ces deux logiciels sont distribués gratuitement. Apache est le serveur Web le plus répandu et fonctionne sur toutes sortes de systèmes d'exploitation. PHP (« PHP : Hypertext Preprocessor ») est un langage de script HTML qui s'exécute côté serveur. Il permet au serveur de générer dynamiquement des pages Web. Un

serveur Web est inutile s'il n'héberge pas de pages Web, c'est pourquoi nous avons conçu un site Web que nous détaillons plus loin.

Afin de supporter les agents mobiles, nous avons installé la plate-forme Grasshopper 2 et le JDK 1.3. De plus, pour pouvoir exécuter notre agent mobile, le serveur dispose des bibliothèques Java *HTMLParser* et *Dynamix*.

5.2.2 Configuration du client

Le client utilisé est un ordinateur portable de la marque Toshiba.

Ressources matérielles

La machine client est constituée d'un microprocesseur Intel III cadencé à 800 MHz, et de 128 Mo de mémoire vive. Elle dispose de deux types de connexion, d'une part, d'une connexion commutée grâce à un modem 56 Kbps, et d'autre part, d'une connexion en réseau local à l'aide d'une carte Fast Ethernet 100 Mbps.

Logiciels installés

Le client fonctionne avec le système d'exploitation Windows 2000 version professionnelle. La machine virtuelle Java (JDK 1.3), ainsi que la plate-forme Grasshopper 2 sont installées. De plus, sont présentes les bibliothèques Java *HTMLParser* et *Dynamix*.

5.2.3 Site test

Afin de comparer efficacement les deux versions de notre application, nous avons créé un site Web dynamique hébergé sur notre serveur. Le site est programmé en PHP et permet ainsi de générer des pages à la demande. Nous avons conçu un site très simple répondant aux exigences de nos différents tests. C'est-à-dire que le site doit contenir un formulaire, générer des pages en fonction des mots-clés des requêtes, mais

aussi ces pages doivent être de longueurs variables afin de comparer nos deux approches sur des données de longueurs différentes.

La page d'accueil de notre site est donc composée d'un formulaire très simple, comme celui de la Figure 4.9. On peut donc formuler une requête. Suivant la requête, nous générons des pages Web différentes. Toutes les pages ont la même apparence, seul le contenu textuel change. En effet, suivant les mots-clés de la requête, nous faisons appel à différents fichiers texte qui vont venir s'intégrer à la page Web. Le Tableau 5.1 présente la longueur des pages générées en fonction des mots-clés entrés. Si la requête est différente de ces mots-clés, la page générée retournera l'information suivante : « pas de résultats ».

Tableau 5.1 Taille des pages du site test

Mots-clés	agent	crawler	sites dynamiques	amélioration	Web invisible
Taille des pages (Ko)	0,6	4,9	46	111	666

Nous avons ainsi un panel étendu de pages de longueurs distinctes. Une page très courte représentera une recherche sur une faible quantité de données, alors qu'une page très longue sera équivalente à une recherche sur une branche de pages contenant une importante quantité de données. Il est essentiel d'avoir ces différentes longueurs de pages, car la longueur des pages devrait influencer beaucoup les exécutions de notre application.

5.3 Expériences

Nous avons réalisé différentes expériences afin d'évaluer notre modèle et de comparer les approches client/serveur et agent mobile, à la lumière des différents critères que sont la charge réseau, la rapidité d'exécution, et la consommation des ressources du serveur. En ce qui concerne la réponse en temps, nous avons réalisé deux séries d'expériences : une première série avec une connexion à faible débit et une deuxième avec une connexion haut débit. Ces deux séries permettent de rendre compte de deux types de connexions distinctes très utilisées pour accéder au Web.

Pour chacun de ces points, nous donnerons le plan d'expériences puis l'évaluation des performances pour nos deux approches, et enfin nous comparerons les résultats obtenus. Il est à noter que, pour toutes les expériences suivantes, nous utilisons les machines client et serveur, ainsi que le site test décrit dans la section précédente. Le détail des expériences est précisé dans les plans d'expérience de chacune des sous-sections suivantes.

5.3.1 Évaluation de la réponse en temps avec une connexion faible débit

La série d'expériences suivante permet de tester notre modèle dans des conditions très réalistes. Aujourd'hui, de nombreux terminaux ont accès au Web avec une connexion de l'ordre de 56 Kbps. Les ordinateurs personnels connectés au moyen d'une liaison commutée traditionnelle utilisent des modems 56 Kbps. Mais surtout, de plus en plus de petits terminaux (téléphones cellulaires, PDA, ...) ont un accès sans-fil de faible capacité au Web.

Plan d'expérience

Le serveur et le client sont tous les deux connectés à Internet, le premier avec une connexion ADSL et le second avec un modem classique. Ainsi, nous serons limités en débit par les 56 Kbps du modem du client. Les deux machines sont connectées à Internet

via des domaines distincts, nous sommes donc dans une situation tout à fait réaliste d'accès au Web.

Pour chacune des deux approches nous effectuerons cinq séries de requêtes sur le site test. Ces séries correspondent à des requêtes générant des pages Web de tailles différentes de manière à évaluer les approches en fonction de la taille des pages Web à analyser.

Nous nous intéressons ici à la réponse en temps. Nous la mesurons tout simplement en utilisant dans le code Java, la méthode *System.currentTimeMillis()* qui retourne le temps de l'horloge de la machine. Nous avons réalisé un script afin d'automatiser les requêtes et d'écrire les durées d'exécution sur un fichier de résultat.

L'état de congestion du réseau pouvant influencer sur les résultats, nous avons choisi de répéter un grand nombre de fois les requêtes au serveur. Cependant, la dernière série de requêtes génère une page Web de très grande taille. La connexion étant à faible débit, la durée d'exécution de l'approche client/serveur est de quelques minutes. Pour avoir une durée d'expérience de quelques heures, nous avons limité nos séries de requêtes à 40 exécutions. Ceci est très raisonnable puisque nous allons voir que les écarts-types sont faibles.

Approche Client/Serveur

Les résultats de l'exécution de l'approche Client/Serveur sont présentés au Tableau 5.2 et à la Figure 5.2.

Tableau 5.2 Réponse en temps de la version client/serveur faible débit (56 Kbps)

Taille de la page (Ko)	0,6	4,9	46	111	666
temps moyen (ms)	1965,825	3561,625	19104,975	42878,425	247783,8
écart-type(ms)	154,098165	238,963144	94,9967072	558,220516	2141,85596

Nous constatons que la durée d'exécution augmente proportionnellement avec la taille de la page à analyser. L'écart type est négligeable par rapport à la durée totale.

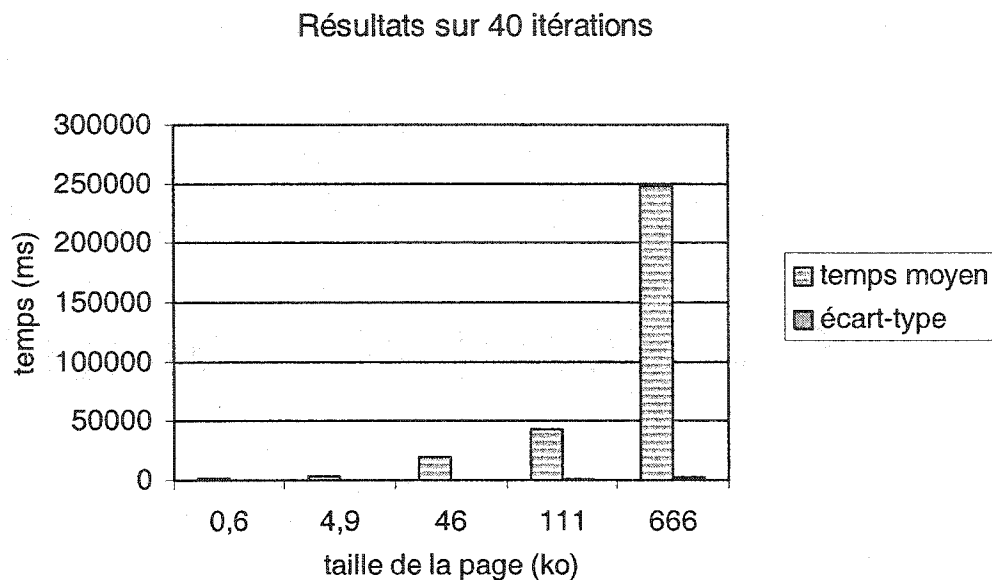


Figure 5.2 Réponse en temps de la version client/serveur faible débit (56 Kbps)

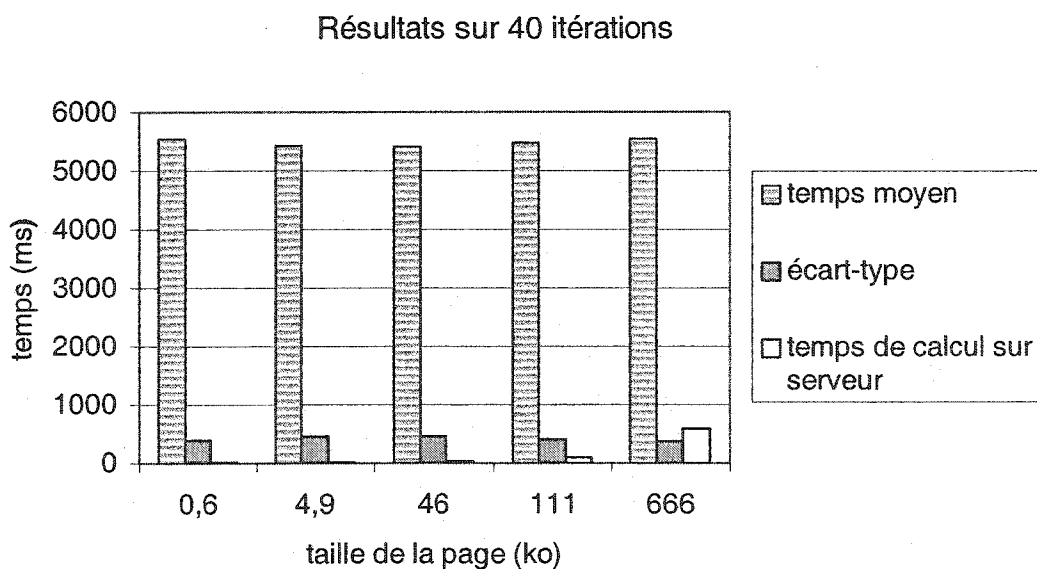


Figure 5.3 Réponse en temps de la version agent mobile faible débit (56 Kbps)

Approche agent mobile

Les résultats de l'exécution de l'approche client/serveur sont présentés au Tableau 5.3 et à la Figure 5.3.

Tableau 5.3 Réponse en temps de la version agent mobile faible débit (56 Kbps)

taille (Ko)	0,6	4,9	46	111	666
temps moyen(ms)	5545,74359	5433,20513	5417,28205	5475,30769	5541,94872
écart-type(ms)	399,706926	453,342299	454,365827	414,965563	374,39003
temps de calcul sur serveur(ms)	20	20	40	100	590

La durée d'exécution est à peu près constante suivant les pages à analyser, et l'écart type reste faible. Seule la durée d'exécution sur le serveur dépend de la taille de la page à analyser, mais elle est négligeable devant la durée de transfert de l'agent mobile.

Comparaison : un avantage certain à l'approche agent mobile

La Figure 5.4 compare les temps d'exécution des deux approches. Conformément à nos attentes, l'approche agent mobile a une durée d'exécution à peu près constante, alors que l'approche client/serveur a une durée d'exécution proportionnelle à la taille des données à traiter. En effet, comme on a une liaison à faible débit, l'essentiel de la durée de la recherche est dû au délai de transfert des données. Dans le cas de l'approche agent mobile, il s'agit de transférer l'agent, et dans le cas de l'approche client/serveur, on transfère toutes les données à analyser. Ainsi, l'approche agent mobile devient intéressante dès que la taille des données à analyser devient plus importante que la taille de l'agent mobile. Sur le graphique, on constate que cette limite se situe aux alentours de 10 Ko. Or, une page Web a une taille de plusieurs dizaines de Ko, et une recherche analyse sur un site Web dynamique en général plusieurs, voire plusieurs dizaines de pages. Pour 100 Ko de données à analyser, la version agent mobile

effectue la recherche en 5.5 secondes alors que la version client/serveur met 38 secondes, soit environ sept fois plus de temps. Ainsi, lorsque l'on dispose d'une connexion Internet à faible débit, il est beaucoup plus avantageux de procéder à une recherche dynamique au moyen d'agent mobile.

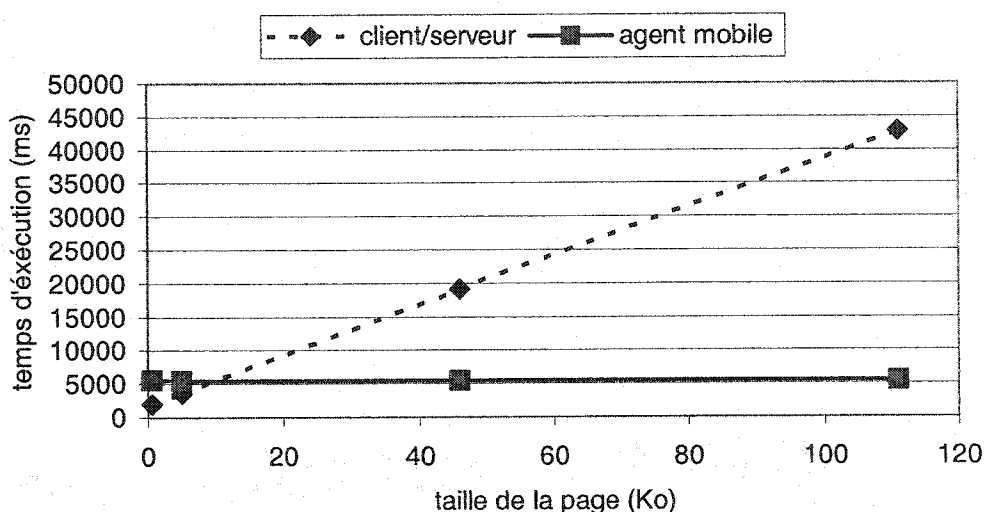


Figure 5.4 Comparaison des temps de réponse (faible débit)

5.3.2 Évaluation de la réponse en temps avec une connexion haut débit

De plus en plus de machines sont connectées à des réseaux locaux et/ou à Internet à l'aide de liaisons haut débit de l'ordre de 100 Mbps, notamment par l'intermédiaire de réseaux privés ou académiques. Mais aussi de nombreux particuliers disposent de connexions par câble ou par la technologie ADSL, connexions qui atteignent quelques Mbps. Dans cette sous-section, nous étudions les réponses en temps de notre recherche dynamique dans le cas d'une liaison haut débit.

Plan d'expérience

Les machines client et serveur sont reliées entre elles directement au moyen de leurs cartes réseau Fast Ethernet et d'un câble croisé RJ45. Ceci permet de connecter les deux machines dans un réseau local à 100 Mbps.

Nous allons procéder exactement comme pour les expériences avec une connexion à faible débit. Mais, comme nous sommes beaucoup moins limités par le temps de transfert des données, nous pouvons reproduire les tests un plus grand nombre de fois. Ainsi, nous ferons des séries de 100 recherches pour avoir des données suffisamment significatives.

Approche client/serveur

Les résultats de l'exécution de la recherche selon l'approche Client/Serveur sont présentés à la Figure 5.5.

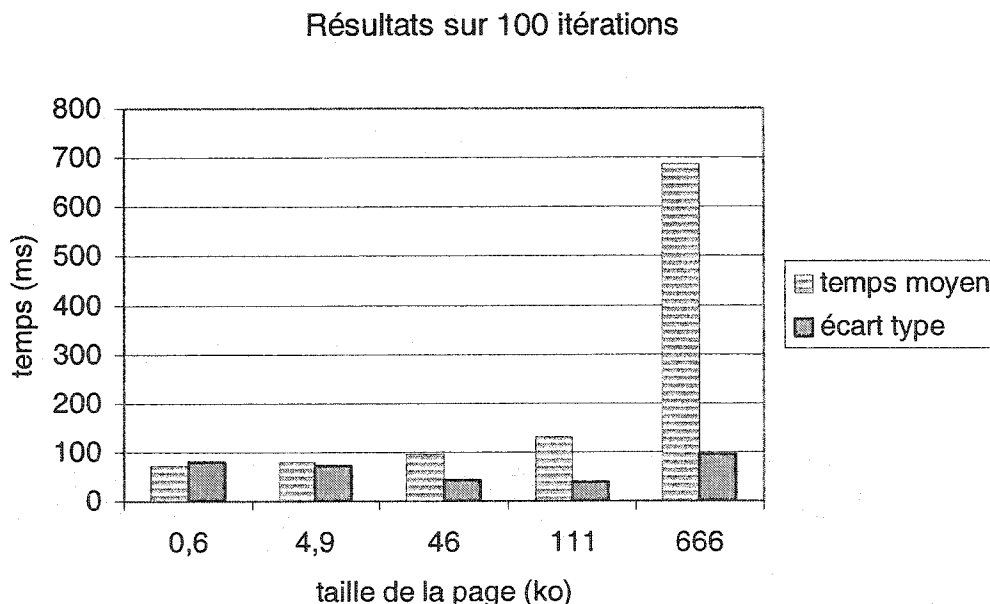


Figure 5.5 Réponse en temps de la version client/serveur haut débit (100 Mbps)

Comme dans les expériences en faible débit, nous constatons que la durée de recherche augmente avec la taille des données à analyser. Cependant, la recherche est beaucoup plus rapide cette fois-ci. Pour une page de 111 Ko, la recherche est réalisée en 130 ms contre plus de 42 s dans les expériences en faible débit, soit une recherche plus de 300 fois plus rapide.

Nous constatons que bien qu'inférieurs en valeurs absolues à ceux trouvés dans les expériences précédentes, les écarts types sont assez élevés relativement aux durées moyennes des recherches. Les délais de transmission des données sont assez stables et influent peu sur l'écart type. Par contre, les temps de traitement sur les machines fluctuent beaucoup plus pour des raisons sur lesquelles nous ne nous sommes pas attardés. Comme entre nos deux séries d'expériences (faible débit et haut débit), seul le temps de transmission varie et qu'il est assez stable, il est normal d'avoir un rapport écart type sur temps moyen plus important dans le cas haut débit.

Approche agent mobile

Les résultats de l'exécution de la recherche selon l'approche agent mobile sont présentés à la Figure 5.6. Comme dans la série d'expériences précédente, la durée de la recherche dépend très peu de la taille des pages à analyser. On peut supposer le temps de recherche constant. La durée de recherche a diminué de plus de moitié par rapport au faible débit. Le temps de transfert de l'agent étant négligeable, et la durée d'exécution ne dépendant pas de la taille des données, on peut supposer que le délai de la recherche est essentiellement causé par le temps de création de l'agent.

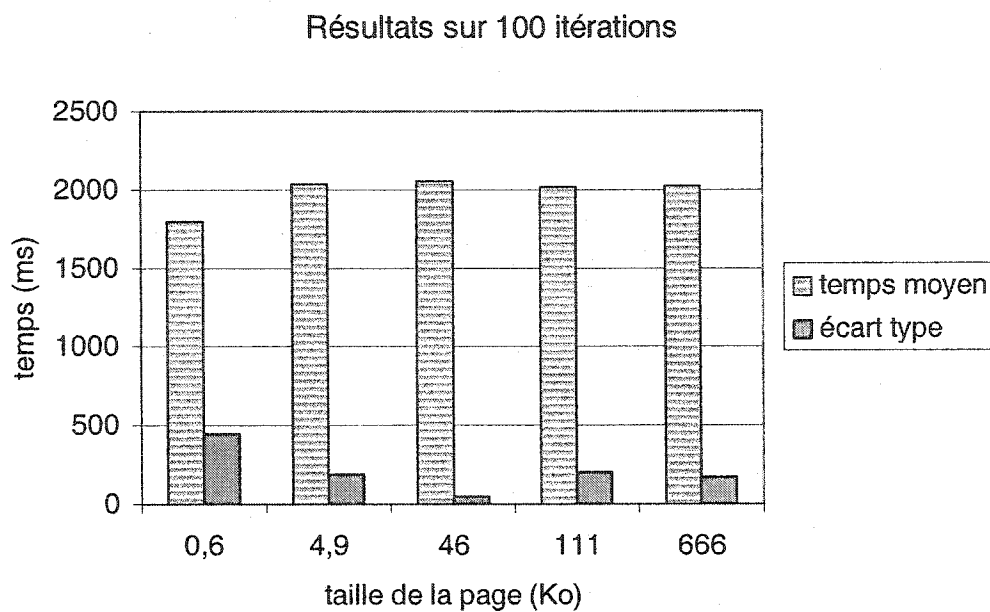


Figure 5.6 Réponse en temps de la version agent mobile haut débit (100 Mbps)

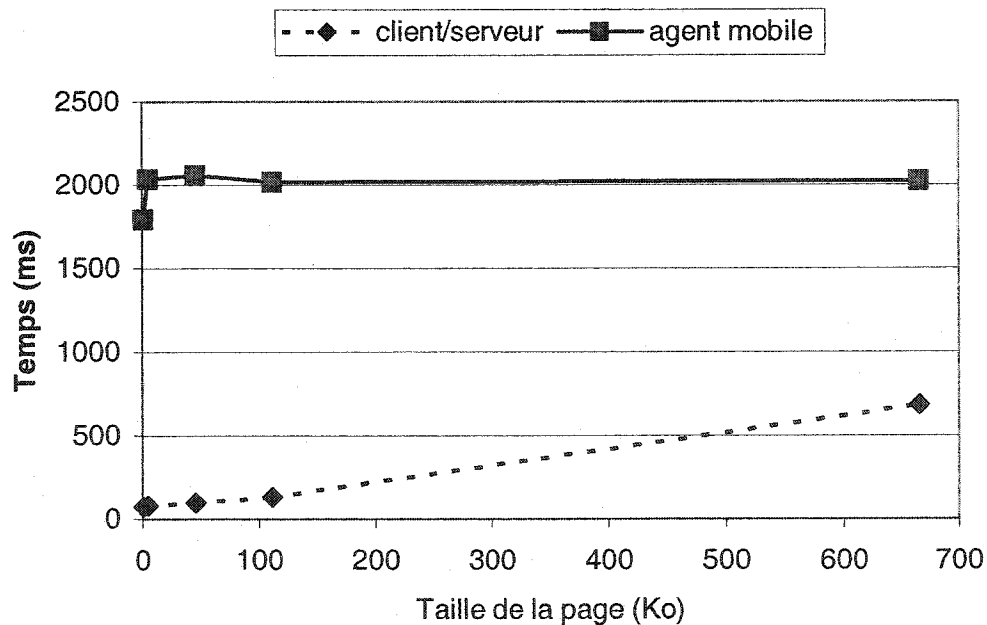


Figure 5.7 Comparaison des temps de réponse (haut débit)

Comparaison : avantage à l'approche client/serveur

La Figure 5.7 compare les temps d'exécution de la recherche selon les deux approches. Nous constatons ici que, même pour des volumes d'information assez gros, l'approche client/serveur est plus rapide. Nous pouvons supposer que, lorsque nous effectuons une requête sur un site Web dynamique, il nous renvoie en général des quantités d'information textuelle n'excédant pas quelques Mo. Pour 500 Ko de données à analyser, l'approche agent mobile (2 s) est quatre fois plus lente que l'approche client/serveur (500 ms). De plus, pour des pages de taille faible, le temps de recherche est négligeable pour l'approche client/serveur, alors qu'il est toujours de 2 s pour l'approche agent mobile.

5.3.3 Évaluation du trafic généré

Lors des deux séries de mesures précédentes, nous nous sommes intéressées à la rapidité d'exécution de la recherche dynamique. Cependant, ce n'est pas le seul critère important qui entre en compte dans l'évaluation comparative de nos deux modèles. En effet, la quantité de trafic généré est un élément crucial. L'application développée a pour vocation une utilisation intensive (pour information, Google traitait plus de 130 millions de requêtes par jour en 2002) . Ainsi, si elle est trop gourmande en trafic, elle peut vite saturer de nombreuses liaisons.

Plan d'expérience

Pour cette série d'expériences, le débit de connexion importe peu puisque le trafic généré par notre recherche n'en dépend pas du tout. Nous avons choisi de travailler sous haut débit. Nous reprenons ainsi la même configuration que pour la série d'expériences précédente. D'une part, cette configuration est en réseau isolé, ce qui empêche des perturbations des mesures, et d'autre part, le haut débit permet de réaliser les mesures plus rapidement. Vu que la quantité de trafic généré est extrêmement stable,

nous n'avons renouvelé nos mesures que quelques fois, et il s'est avéré inutile de présenter la moyenne d'un nombre de mesures répétées.

Nous avons mesuré les quantités de trafic généré à l'aide du logiciel d'analyse de trafic *EtherPeek*⁷. En sélectionnant les filtres pertinents, il permet d'isoler aisément les données concernant le trafic qui nous intéresse. Pour chacune de nos deux approches, nous avons ainsi mesuré le trafic généré dans le sens client vers serveur, ainsi que dans le sens opposé, et ce, pour un panel pertinent de pages de tailles différentes.

Approche client/serveur

Les résultats de l'exécution de la recherche selon l'approche Client/Serveur sont présentés à la Figure 5.8.

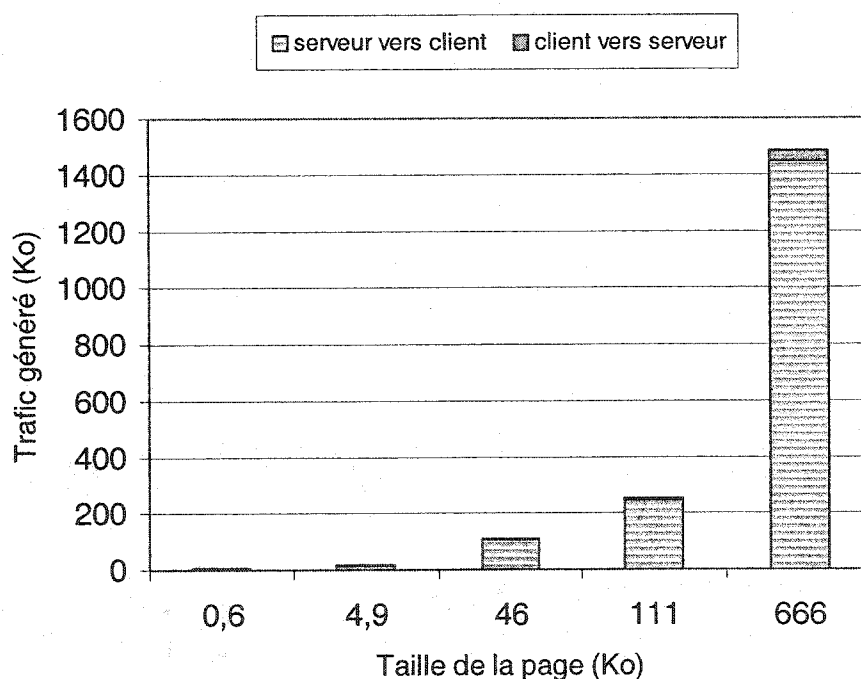


Figure 5.8 Trafic généré par l'approche client/serveur

⁷URL : <http://www.wildpackets.com>

Nous remarquons que le trafic est proportionnel à la taille de la page analysée. Ceci est conforme à nos attentes, puisque, dans l'approche client/serveur, on transfère vers le client la totalité des données à analyser.

Nous constatons que les transferts du client vers le serveur sont négligeables devant les transferts en sens inverse. En effet, dans ce sens, on ne transmet que les requêtes HTTP, le reste des données transférées sont des paquets de contrôle du protocole TCP/IP. La quantité de ces paquets de contrôle est proportionnelle à la quantité d'informations transférées du serveur vers le client.

Le trafic circulant du serveur vers le client est légèrement supérieur à deux fois la taille de la page analysée. En mesurant les détails de l'exécution de la recherche, nous nous sommes rendus compte que notre application téléchargeait deux fois la page à analyser. Ceci est dû à l'utilisation de la bibliothèque *HTMLParser* que nous n'avons pas optimisée. Cependant, ceci ne fait que multiplier les temps de téléchargement par deux et n'affecte pas la signification des résultats. Même si on peut réduire de moitié le trafic, le temps d'exécution sera toujours moins avantageux par rapport à l'approche agent mobile, comme dans le cas de la Figure 5.4. L'essentiel du trafic généré est donc constitué par les données à télécharger, le reste étant des bits de contrôle.

Approche agent mobile

Les résultats de l'exécution de la recherche selon l'approche agent mobile sont présentés à la Figure 5.9. Nous constatons que les quantités de trafic générées restent constantes suivant la taille de la page à analyser. Nous constatons aussi que les transferts du client vers le serveur sont supérieurs de 50% à ceux du serveur vers le client. La recherche dynamique nécessite en effet deux transferts. Tout d'abord, on transfère entièrement l'agent mobile vers le serveur, puis on le fait revenir sur le client. Cependant, Grasshopper ne retransfère pas totalement l'agent mobile car certaines données de l'agent mobile ne changent pas, et peuvent ainsi être gardées en cache. Ceci

explique la différence entre le trafic montant et le trafic descendant. Une certaine proportion du trafic est bien entendue utilisée par les paquets de contrôle.

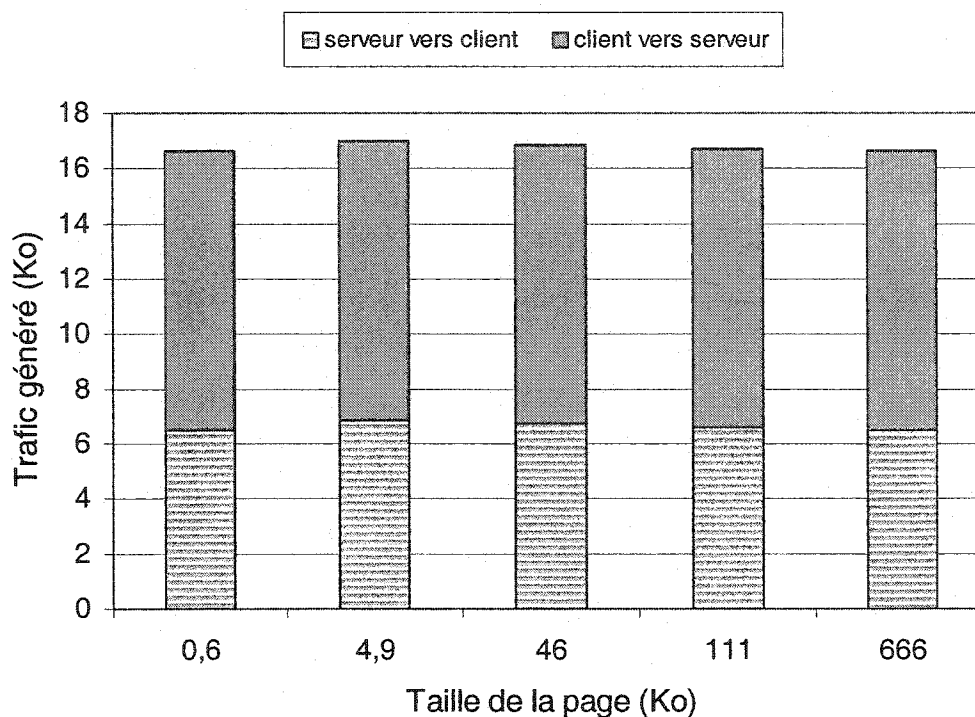


Figure 5.9 Trafic généré par l'approche agent mobile

Comparaison : avantage à l'approche agent mobile

La Figure 5.10 compare les trafics générés par la recherche selon les deux approches. Nous constatons que ce graphique est très similaire à celui de la Figure 5.4 sur les temps de réponse dans le cas d'une liaison faible débit. En effet, dans le cas d'une liaison à faible débit les temps d'exécution de la recherche reflètent essentiellement les quantités de trafic générées.

Nous pouvons tirer les mêmes conclusions que lors de la première série d'expérience. C'est à dire que l'approche agent mobile est avantageuse dès une

recherche sur pages légères, et devient très avantageuse dans le cas de pages plus lourdes.

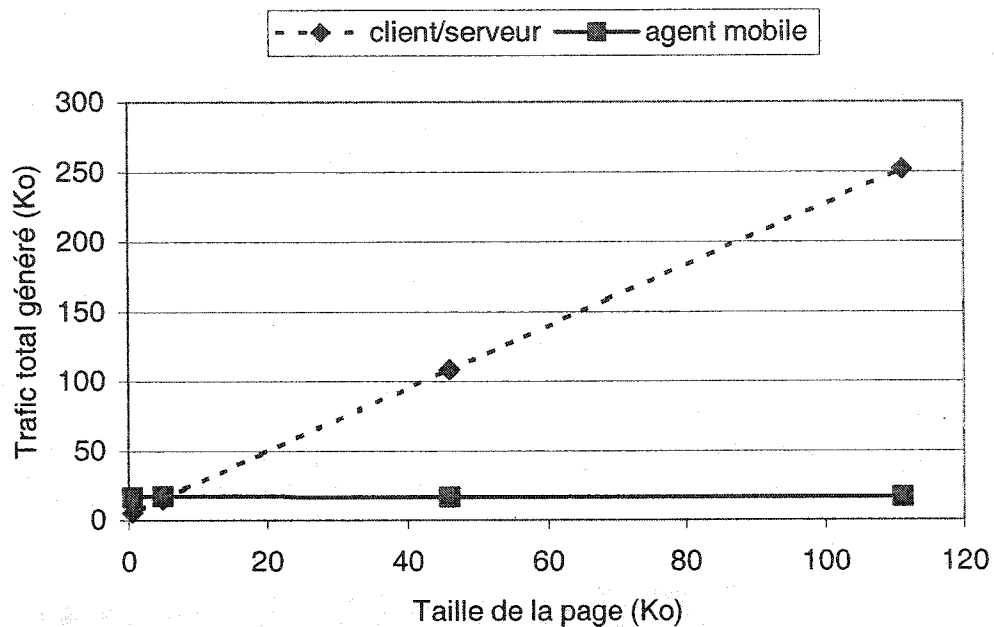


Figure 5.10 Comparaison des trafics générés

Ces résultats sur le trafic viennent modérer les conclusions tirées de la comparaison en temps des approches client/serveur et agent mobile dans le cas d'une liaison haut débit. En effet, même si dans ce cas, l'approche client/serveur est plus rapide, elle génère beaucoup plus de trafic, ce qui peut constituer un désavantage.

5.3.4 Évaluation de la consommation des ressources des machines

Jusqu'à présent, nous nous sommes concentrés sur l'étude des temps de réponse et de la charge réseau engendrés par la recherche dynamique. Le dernier point que nous estimons être important dans la comparaison des deux approches client/serveur et agent mobile, est la consommation des ressources du serveur et du client, c'est-à-dire l'utilisation du processeur et de la mémoire vive par notre application. En effet, un des

avantages des agents mobiles est de consommer moins de ressources du client, mais il ne faut cependant pas qu'il surcharge le serveur.

Plan d'expérience

Dans cette série d'expériences, nous ne nous intéressons ni au trafic généré ni au temps de réponse, mais plutôt aux charges de calcul supportées par le client et par le serveur. Nous avons repris la même configuration que lors des deux expériences précédentes, c'est-à-dire les deux ordinateurs reliés en réseau local.

Nous avons effectué nos mesures à l'aide du gestionnaire des tâches de Windows 2000 qui fournit un historique d'utilisation du processeur et de la mémoire vive. Ainsi, pour chacune de nos deux approches, nous avons mesuré sur les machines client et serveur, les taux d'utilisation du processeur et de la mémoire vive pour des recherches successives sur nos pages test.

Nous avons constaté que l'utilisation de la mémoire vive n'était pas influencée de manière perceptible par notre application. Nous n'avons donc pas jugé utile de présenter ces mesures.

Nous tenons à faire remarquer que les résultats de ces mesures dépendent des ressources des machines que nous avons utilisées pour ces expériences.

Approche client/serveur

La Figure 5.11 présente les résultats des mesures de l'utilisation des processeurs des machines client et serveur pour notre approche client/serveur. Le taux d'utilisation des processeurs augmente légèrement avec la taille des pages à analyser. Le processeur de l'ordinateur serveur est très peu sollicité, l'essentiel du traitement ayant lieu sur le client. Le taux d'utilisation du processeur du client connaît des pics allant de 15% à 40%.

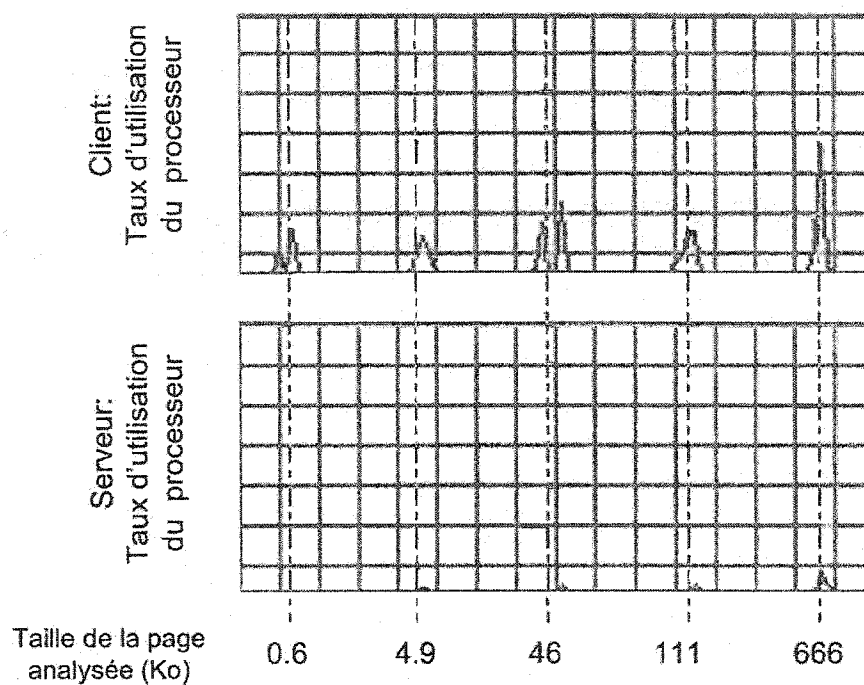


Figure 5.11 Utilisation du processeur dans l'approche client/serveur

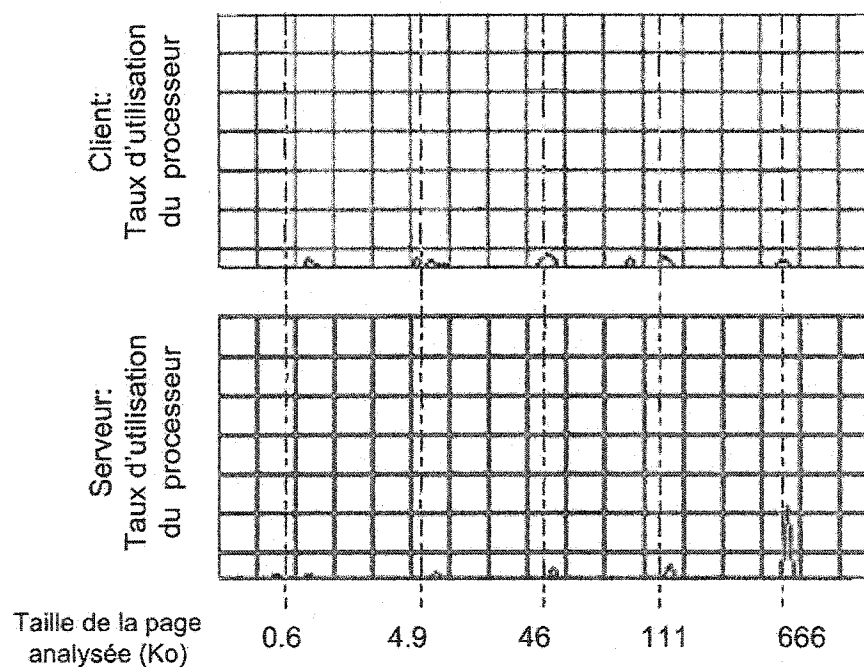


Figure 5.12 Utilisation du processeur dans l'approche agent mobile

Approche agent mobile

La Figure 5.12 présente les résultats des mesures de l'utilisation des processeurs des machines client et serveur pour notre approche agent mobile. Nous constatons que le processeur du client est peu sollicité et que son utilisation ne dépend pas de la taille de la page à analyser. Le processeur du serveur est légèrement plus sollicité et son taux d'utilisation augmente sensiblement avec la taille de la page à analyser.

Comparaison : avantage à l'approche agent mobile pour un client à faibles ressources de calcul

Conformément à nos attentes, l'approche client/serveur sollicite plus le processeur du client, et l'approche agent mobile plus celui du serveur. La diminution de l'utilisation du processeur de la machine client, lorsque l'on passe d'une approche client/serveur à une approche agent mobile paraît appréciable. On passe de taux d'utilisation de l'ordre de 20% à des taux de l'ordre de 7%. Une approche agent mobile peut donc s'avérer très avantageuse pour des clients ayant des capacités de calcul très limitées comme les PDA.

La différence d'utilisation du processeur du serveur est moins nette que pour le processeur du client. On pourrait donc penser que globalement l'utilisation de l'approche agent mobile est avantageuse. Mais cette conclusion est un peu hâtive, il faudrait tester l'influence sur le serveur de plusieurs connexions simultanées, afin de voir si l'approche agent mobile provoque des risques beaucoup plus importants de surcharge du serveur.

5.4 Synthèse des performances

Nous avons effectué quatre séries de test sur les performances comparatives des approches client/serveur et agent mobile de notre modèle de recherche dynamique. Il s'agit maintenant de faire la synthèse de ces différents résultats.

L'approche agent mobile prend un délai de création de l'agent et génère un trafic constant, alors que l'approche client/serveur nécessite le téléchargement de toutes les données sur le client. Ainsi, le temps de réponse est constant pour l'approche agent mobile et il est proportionnel à la quantité de données à analyser pour l'approche client/serveur. Dans le cas d'une liaison à faible débit, l'avantage de l'approche agent mobile va apparaître clairement. Dans le cas d'une liaison à haut débit, l'approche client/serveur sera plus rapide, mais cet avantage est à modérer car cette approche produit nettement plus de trafic.

Du point de vue de la consommation des ressources système, l'approche agent mobile soulage beaucoup l'utilisation du processeur de la machine client, et ne surcharge pas trop le serveur. Néanmoins, il faudrait effectuer des tests avec de nombreux accès simultanés au serveur pour vérifier cette seconde proposition.

Ces résultats sont conformes à nos attentes car ils confirment les avantages attendus des agents mobiles en recherche d'information. Le traitement des informations à la source soulage le réseau et le client, et plus la quantité d'information à traiter est grande, plus cet avantage est grand. De plus, l'approche agent mobile convient particulièrement bien aux clients disposant de peu de puissance de calcul et d'une connexion à faible débit. De ce fait, elle s'adapterait bien à l'informatique mobile.

CHAPITRE VI

CONCLUSION

Dans ce mémoire, nous nous sommes intéressés à la recherche d'information sur le Web invisible. Nous avons élaboré un modèle permettant d'effectuer dynamiquement des recherches sur le Web invisible et nous l'avons implémenté suivant deux versions : client/serveur et agent mobile. Ceci nous a permis d'évaluer les performances comparatives de ces deux approches. Cependant, nous avons limité notre implémentation à une partie de l'architecture globale, et nous n'avons pas effectué d'expériences avec plusieurs clients et avec des recherches sur plusieurs sites à la fois.

Ce chapitre de conclusion va donc nous permettre de faire la synthèse des travaux réalisés, d'expliquer nos limitations et d'indiquer les orientations des recherches futures.

6.1 Synthèse des travaux et contributions

Nous avons élaboré un modèle d'architecture se basant sur un moteur de recherche et permettant d'effectuer en temps réel des recherches sur des sites Web dynamiques. Notre modèle de recherche repose sur deux idées fondamentales. Premièrement, le moteur de recherche peut indexer des URLs de sites dynamiques et les mettre en relation avec des mots-clés lors de son parcours du Web. Ainsi, le moteur de recherche peut nous fournir des listes de liens vers des sites dynamiques adaptés à chaque requête. Deuxièmement, nous effectuons une recherche dynamique sur les sites Web, c'est-à-dire que nous les analysons seulement lorsque l'utilisateur nous adresse sa requête, et non à l'avance lors du crawling comme le font les moteurs de recherche. Ceci nous permet d'accéder au contenu des sites dynamiques, car nous pouvons compléter les formulaires de recherche et générer les pages correspondantes, alors que ce contenu reste *invisible* aux moteurs de recherche.

Notre architecture est souple et peut se mettre en œuvre sans de gros investissements techniques ou matériels. Elle se base en effet sur l'utilisation d'un moteur de recherche préexistant, et ne nécessite pas le stockage des données car la recherche va se faire en temps réel sur les sites Web. L'architecture est souple, notre module de recherche peut aussi bien s'intégrer à un moteur de recherche ou prendre la forme d'un logiciel de recherche installable sur la machine de l'utilisateur. Le but est de fournir à l'utilisateur une facilité de recherche sur le Web invisible comparable à celle des moteurs de recherche sur le Web visible.

La seconde importante contribution de nos travaux est d'avoir développé une application de recherche d'information utilisant la technologie des agents mobiles. En effet, cette technologie manque encore d'applications concrètes. Nous avons pu évaluer les performances de notre application utilisant des agents mobiles en la comparant avec une version utilisant l'approche traditionnelle client/serveur. Les résultats obtenus sont très encourageants et confirment les avantages attendus des agents mobiles. L'utilisation des agents mobiles permet de limiter de façon très significative les transferts sur le réseau et de soulager les ressources de la machine client. En particulier, les agents mobiles sont très avantageux pour des machines client de faible capacité de calcul et à la connexion au réseau de faible débit.

6.2 Limitations des travaux

Comme nous l'avons rappelé dans la section précédente, notre architecture repose sur celle des moteurs de recherche. Pour notre implémentation, nous nous sommes limités à toute la partie qui ne concerne pas le moteur de recherche. En effet, nous aurions dépassé le cadre de ce mémoire de maîtrise en allant proposer un nouvel algorithme pour l'indexage effectué par les moteurs de recherche. Nous n'avons donc pas pu tester si les moteurs de recherche pouvaient nous fournir des listes de liens vers des sites dynamiques qui nous permettraient d'obtenir des résultats suffisamment pertinents.

Une autre limitation importante à nos travaux est l'hétérogénéité des sites Web dynamiques. Les formulaires permettant d'effectuer les requêtes qui vont générer des pages dynamiquement sont en effet de types très différents. Les formulaires sont pour la plupart en HTML, mais peuvent aussi utiliser des frames ou du Javascript. Le plus gros problème est le fait qu'il n'y ait pas de norme pour le nom de chacun des champs du formulaire, et il est très difficile pour un agent non humain de les interpréter correctement. Il faudrait une standardisation de ces champs à la manière du langage XML. Devant cette difficulté, nous nous sommes limités à compléter un unique champ dans des formulaires très simples.

Nous tenons à souligner un dernier point concernant les limitations de nos travaux. À l'heure actuelle, les agents mobiles posent encore d'importants problèmes de sécurité [15]. Le problème le plus important est de défendre l'agent mobile contre des serveurs malicieux. Il est vrai qu'en recherche d'information ce problème est moins crucial qu'en commerce électronique, mais quand l'on sait combien il est important pour un site commercial de bien se faire référencer, la sécurité peut être un enjeu important. Quoiqu'il en soit, tant que l'on n'aura pas apporté des solutions satisfaisantes à la sécurité des agents mobiles, on ne pourra pas mettre en œuvre sérieusement à l'échelle de l'Internet des applications à base d'agents mobiles.

La conséquence directe de ce dernier point est que nous n'avons pas pu tester notre modèle de recherche par agents mobiles sur des sites opérationnels. Nous avons dû nous contenter d'utiliser un site supportant les agents mobiles, conçu spécialement pour nos tests.

6.3 Orientations de recherches futures

Nous avons comparé les performances des approches client/serveur et par agents mobiles seulement lors de recherches sur un site Web unique. Il serait très intéressant d'étendre cette comparaison à des recherches sur plusieurs sites en même temps. Ces tests complémentaires seraient certainement à l'avantage des agents mobiles.

L'approche par agents mobiles permettrait en effet une parallélisation des tâches bien meilleure que l'approche client/serveur. Le scénario développé à la Figure 3.3 explique bien la recherche multi-sites dans le cas de l'utilisation d'agents mobiles. Dans le cas d'une approche client/serveur, une recherche sur plusieurs sites à la fois multiplierait les calculs à effectuer par le client ainsi que la quantité des transferts à réaliser.

La prochaine étape pour valider complètement ce modèle de recherche serait de s'intéresser directement à l'indexage des moteurs de recherche. Nous pourrions alors tester globalement notre architecture. Le point le plus important à surveiller serait la pertinence des résultats des recherches, ce qui constitue, avec le délai de réponse, le critère le plus important pour évaluer un outil de recherche.

Enfin, une amélioration possible de notre modèle de recherche serait de personnaliser nos agents mobiles et de les rendre évolutifs afin qu'ils s'adaptent aux différents sites Web visités. Nous pourrions réaliser une bibliothèque évolutive d'agents mobiles classés en fonction des caractéristiques de sites Web à visiter, et suivant les sites fréquemment visités. Une telle personnalisation permettrait d'effectuer des recherches plus puissantes sur les sites dynamiques en exploitant tous leurs champs de recherche.

BIBLIOGRAPHIE

- [1] A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke et S. Raghavan, « Searching the web », *ACM Transactions on Internet Technology*, 2001, pp. 1-42.
- [2] M. K. Bergman, « The deep web : Surfacing hidden value », *The Journal of Electronic Publishing*, Vol. 7, No. 1, Août 2001.
- [3] F. Bourdoncle, « Panorama et perspectives des outils de recherche d'information textuelle sur Internet », In *IDT/NET'99*, Paris, 8-10 juin 1999.
- [4] O. Andrieu, *Abondance*, [http ://www.abondance.com](http://www.abondance.com), 2002.
- [5] F. Bourdoncle, *Exalead*, [http ://www.exaleadcom](http://www.exaleadcom), 2002.
- [6] A. Broder, R.Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, et J. Wiener, « Graph structure in the Web », *Computer Networks*, Vol. 33, No. 1-6, Juin 2000, pp. 309-320.
- [7] S. Lawrence et C. Lee Giles, « Accessibility of information on the Web », *Nature*, Vol. 400, juillet 1999, pp. 107-109.
- [8] J. Cho et H. Garcia-Molina, « Synchronizing a database to improve freshness », In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, Dallas, 16-18 mai 2000, pp. 117-128.

- [9] B. Brewington et G. Cybenko, « How dynamic is the web ? », *WWW9 / Computer Networks*, Vol. 33, No. 1-6, 2000, pp.257-276.
- [10] R. Green et S. Pant, « Multiagent data collection in lycos », *Communications of the ACM*, Vol. 42, No. 3, 1999, p. 70.
- [11] C. Ding, X. He, P. Husbands, H. Zha, et H. D. Simon, « Pagerank, hits and a unified framework for link analysis », In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Tempere, Finlande, 11-15 août 2002, pp. 353-354.
- [12] C. Sherman et G. Price, *The invisible Web : uncovering information sources search engines can't see*, Information Today, Inc., Medford, Etats-Unis, 2001.
- [13] M. Levene et A. Poulovassilis, « Web dynamics », *Software focus*, Vol. 2, No. 2, 2001, pp.60-67.
- [14] M. Greenberg, J. Byington et D. Harper, « Mobile agents and security », *IEEE Communications Magazine*, Vol. 36, No. 7, 1998, pp. 76-85.
- [15] G. Vigna (ed.), *Mobile Agent Security*, Springer-Verlag, Heidelberg, Allemagne, 1998.
- [16] A. Karmouch et V. A. Pham, « Mobile Software Agents : an Overview », *IEEE Communications , Special Issue on Mobile Agents and Telecommunications*, Vol. 36, No. 7, 1998, pp. 26-37.

- [17] D. Milojicic, « Mobile agent applications », *IEEE Concurrency*, Vol. 6, No. 3, 1999, pp. 80-90.

- [18] B. Brewington, R. Gray, K. Moizumi, D. Kotz, G. Cybenko et D. Rus, « Mobile Agents for Distributed Information Retrieval », In Matthias Klusch (ed.), *Intelligent Information Agents*, Springer-Verlag, Heidelberg, Allemagne, 1999, pp. 355-395.