2m11. 2750. 1

Université de Montréal

Enrichissement d'un curriculum par recherche sur Internet

par
Michèle Ouellet

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en informatique

Août 1999

© Michèle Ouellet 1999

QA
76
U54
1999
V.036

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé:

Enrichissement d'un curriculum par recherche sur Internet

présenté par:

Michèle Ouellet

a été évalué par un jury composé des personnes suivantes:

Paul Bratley - Président-rapporteur

Jan Gecsei - Directeur de recherche

Jian-Yun Nie - Codirecteur

Houari Abdelkri Sahraoui - Membre du jury

Mémoire accepté le: 99-10-13

Sommaire

Ce mémoire porte sur la gestion de l'information, plus particulièrement sur la découverte de ressources pertinentes pour un sujet donné ainsi que sur l'organisation subséquente de ces ressources.

La question générale qui a motivé ce projet est la suivante: peut-on imaginer un système qui enrichisse un réseau de concepts de façon autonome, au moyen de recherches sur l'Internet? Est-il certaines parties de la formulation des requêtes et du triage des résultats que l'on puisse automatiser?

Notre objectif immédiat était de construire un outil pouvant aider un concepteur pédagogique à trouver des documents pertinents pour les sujets abordés par un cours et à organiser ces documents: il faudrait trouver des documents, les classer dans une structure de concepts et les retrouver dans cette structure. Il faudrait effectuer ces opérations avec assez d'autonomie pour alléger la tâche de l'usager.

Nous appelons "curriculum" une ressource pédagogique contenant un plan de cours, soit le nom et l'organisation des matières enseignées avec quelques termes descriptifs pour chaque matière. Nous supposons une structure hiérarchique, un arbre, et nous faisons l'hypothèse suivante: pour chaque noeud de l'arbre, on dispose déjà de quelques documents jugés pertinents pour le concept représenté par le noeud. C'est-à-dire que l'on supposera l'existence d'une bibliographie rudimentaire rattachée à chacun des éléments du cours. Cette hypothèse suffit-elle pour construire un système de repérage et de classement de documents?

Pour répondre à cette question, nous avons construit le prototype d'un tel système. Nous l'avons appelé un Classeur actif; c'est un Classeur, non seulement parce qu'il permet d'organiser des ressources mais aussi parce qu'il est à même de faire une partie du traitement lui-même; c'est un Classeur actif parce qu'il interroge des moteurs de recherche d'information pour aller cueillir de nouvelles ressources pertinentes. Nous avons écrit ce prototype en Java et nous l'avons testé sur un curriculum préparé pour un cours d'Initiation à Internet à l'Université de Montréal.

Le mémoire commence par un tour d'horizon des outils s'adressant au problème de la gestion de l'information sur Internet. Il décrit aussi les techniques de recherche d'information qui constituent l'ossature du Classeur. Puis, le mémoire détaille la conception et la réalisation du Classeur actif ainsi que les tests effectués. Il expose les limites de l'approche choisie et se termine par une évocation des systèmes de gestion des connaissances vers lesquels doivent évoluer le Classeur actif et les autres outils de médiation de l'accès à l'information.

Table des matières

Liste des tableaux

Liste des figures

Liste des sigles et abréviations

Sigle ou abréviation	Signification
AAAI	American Association for Artificial Intelligence
CGI	Common Gateway Interface
FTP	File Transfer Protocol
HTML	HyperText Markup Language
IDF	Inverse Document Frequency
IRTF	Internet Research Task Force
IU	Interface usager
KADS	Knowledge Acquisition and Design System
OO	Object Oriented
RI	Recherche (ou repérage) d'information
SBC	Système à base de connaissances
SOIF	Summary Object Interchange Format
SRI	Système de recherche (ou repérage) d'information
TLN	Traitement langue naturelle
URL	Uniform Resource Locator

À Keith et Alexandre Reuben

Je remercie mon directeur de recherche, le professeur Jan Gecsei, de m'avoir intéressée à ce sujet. Je remercie mon co-directeur, le professeur Jian-Yun Nie, de m'avoir initiée à la RI. Ils ont fait montre de beaucoup de patience, de gentillesse et d'esprit critique. Je les remercie vivement pour la qualité de leurs interventions tout au long de la rédaction de ce mémoire.

Je remercie tous mes amis de chez Stelvio; l'environnement que nous avons créé là-bas, au fil des années, représente un idéal de qualité de logiciel et de qualité de vie que je souhaite à tous ceux qui travaillent dans le domaine de l'informatique.

Chapitre I: La gestion d'information sur Internet

Soudain la quantité d'information disponible à travers un simple micro-ordinateur relié à l'Internet est inimaginable, ce qui fait que cette information reste largement ignorée ou sous-utilisée. De plus, cette information ne cesse d'évoluer, de grandir ou même de vieillir. La gestion d'information est un vieux problème, exacerbé à l'heure actuelle par le rythme de croissance des données. Ce terme de gestion d'information englobe bien des champs d'activité, par exemple la définition et la structuration des ressources, leur accès, leur manipulation, et aussi leur cueillette, leur sélection et leur présentation. Notre travail se situe dans un contexte de filtrage d'information et s'adresse plutôt aux trois dernières facettes.

Pour pouvoir accéder à l'information, pour la trouver, la retrouver et l'utiliser de façon efficace, les outils manquent.

Pour trouver l'information, l'usager doit chercher en passant par une interface qui lui permettra de décrire son besoin. Du point de vue du type de description que l'usager devra fournir, on peut distinguer deux cas:

- description logique faite à un niveau conceptuel, par exemple "langage de publication sur le Web";
- description physique faite à un niveau textuel ou au niveau des champs d'une base de données; chercher suppose alors une connaissance intime des objets recherchés, par exemple, "PostScript encapsulé version 3.0" ou encore un SELECT dans une base de données.

Parmi l'outillage utilisable pour trouver l'information sur Internet, on compte les systèmes d'interrogation des bases de données, les systèmes de repérage d'information (SRI), les systèmes question-réponse, les bibliothèques numériques et enfin les systèmes de navigation et de visualisation de l'information. Certaines de ces interfaces, comme les systèmes d'interrogation de bases de données et les systèmes question-réponse, présupposent que l'information sous-jacente est fortement structurée, qu'elle possède une structure intelligible pour la machine. Or actuellement, la plus grande partie de l'information ne présente aucune structure au-delà d'une utilisation minimale des balises en HyperText Markup Language (HTML). On assiste par contre à la naissance des outils d'extraction d'information; ces outils, dont nous reparlerons au chapitre VII, exploitent diverses connaissances pour interpréter l'information dans certains domaines ponctuels: ils sont ensuite à même de répondre à des questions. Mais ces systèmes sont actuellement très spécialisés et peu déployés. C'est ce qui rend les SRI si importants, surtout dans leur

incarnation plein-texte. Bref, les outils pour trouver l'information ne manquent pas, mais ils sont généralement lourds et peu efficaces.

Supposons l'information trouvée: l'usager a découvert une ressource intéressante et il a créé un signet ou une copie locale. *À la longue, cet univers d'information personnel devient complexe et il devient difficile de s'y orienter. Pour retrouver l'information, l'usager doit encore chercher, mais cette fois dans un univers plus restreint qui est celui des documents et signets déjà jugés intéressants par lui ou par son groupe de travail.* Ici encore, avec un système d'exploitation moderne, on a diverses façons de procéder:

- par navigation, dans une organisation plus ou moins hiérarchique des documents;
- par interrogation sur le titre des documents;
- par interrogation sur les documents ayant certaines méta-données ou contenant certaines chaînes.

Mais cette information, pourquoi la cherchions-nous et qu'allons-nous en faire? Actuellement, le terme de gestion d'information ne couvre pas encore l'aspect utilisation et se concentre sur la localisation et l'accès. Bientôt, les systèmes iront beaucoup plus loin, à la fois en largeur et en profondeur. Un pionnier de la visualisation de l'information, Stuart K. Card, dresse dans [Card96] une taxonomie des outils d'accès à l'information: au plus bas niveau, il y a le document; un peu plus haut, les "sense-making tools", ce que nous pourrions traduire par outils d'appréhension d'information; encore plus haut, l'espace de travail et enfin l'infosphère. Les outils d'appréhension, ces outils permettant d'envisager de grandes quantités d'information, seront bientôt disponibles dans nos environnements.

Le physicien Richard Feynman faisait un jour observer que l'oeil humain est plutôt remarquable comme filtre d'information que comme collecteur d'information [Avis97]. Avec la visualisation, l'oeil est mis à contribution de manière autre que la saisie linéaire favorisée par le texte depuis tant de générations. Mais la visualisation n'est qu'un outil au service de la gestion d'information. Et la gestion d'information n'est pas encore la gestion des connaissances: les deux dernières couches de la taxonomie de Card, l'espace de travail et l'infosphère, relèvent plutôt de ce dernier type de gestion. Nous reviendrons à la gestion des connaissances au dernier chapitre, pour décrire maintenant le problème qui fait l'objet de ce mémoire.

1.1 Sous-problème choisi

Parmi les problèmes liés à la gestion d'information sur Internet, ce mémoire fait abstraction des questions d'accès et d'utilisation pour se concentrer sur le sous-problème consistant à trouver et

retrouver l'information, autrement dit, le problème des requêtes et le problème du classement. La figure 1-1 ci-dessous illustre notre objectif: on y voit trois usagers aux prises avec de grandes quantités d'information. Le premier ne dispose d'aucun outil outre son navigateur, à travers lequel il invoque un SRI pour trouver les ressources qui l'intéressent. Il doit consacrer beaucoup de temps à formuler les requêtes de façon à bien cerner son sujet; puis il doit encore passer beaucoup de temps à explorer les URL retournés pour décider s'ils l'intéressent vraiment. À la fin, il dispose d'une masse de documents peu organisée.

Le deuxième usager, dans la figure, peut déjà travailler un peu plus efficacement. Il dispose d'un outil de requêtes qui fonctionne de manière semi-autonome; l'usager spécifie son besoin d'information à un niveau plus conceptuel; il peut indiquer les sites qui l'intéressent, les sites à éviter; il peut aussi demander que plusieurs SRI soient interrogés et que les opérations aient lieu plus tard, à une heure où le réseau est moins chargé. Si l'on suppose une interaction prolongée et une capacité d'apprentissage de la part de l'outil de requêtes, cet outil pourra même faire un certain filtrage des URL et diminuer d'autant le travail requis de l'usager. Mais au bout du compte, l'usager doit encore travailler avec une masse de documents peu organisée; tout de même, il aura eu moins de mal à les obtenir.

Notre objectif est d'en arriver à l'environnement de travail du troisième usager. Cet usager a externalisé son modèle conceptuel du sujet qui l'intéresse. Il peut encore traiter avec son agent de requêtes mais il peut tout aussi bien laisser ce dernier exploiter le modèle conceptuel directement. Outre la structure de concepts externe, la figure révèle un autre élément nouveau, un agent de classement: ce dernier a pour tâche d'intégrer les ressources repérées par l'agent de requêtes dans la structure fournie par l'usager ou de les rejeter comme non pertinentes à la structure de concepts en question. Il pourrait même suggérer la fusion ou la fission de concepts le cas échéant (voir chapitre VII).

Il pourrait sembler que l'effort requis pour créer une structure de concepts externe, toute rudimentaire qu'elle soit, n'est pas à la portée d'un usager quelconque. Il est vrai que les méthodes constructivistes [Eklu95] semblent avoir peu d'impact sur l'éducation dispensée généralement dans les écoles. Mais supposons que notre usager soit un travailleur intellectuel, par exemple un professeur; ce professeur donne un cours, il a donc dû construire un plan de cours explicite, ce que nous appellerons un *curriculum*. Un plan de cours quelque peu raisonnable devrait révéler quelque chose sur la structure de concepts sous-jacente. Ce curriculum serait aussi assorti d'une bibliographie sommaire. Ce professeur et ses étudiants sont nos usagers-cibles, bien que l'approche soit en fait plus générale.

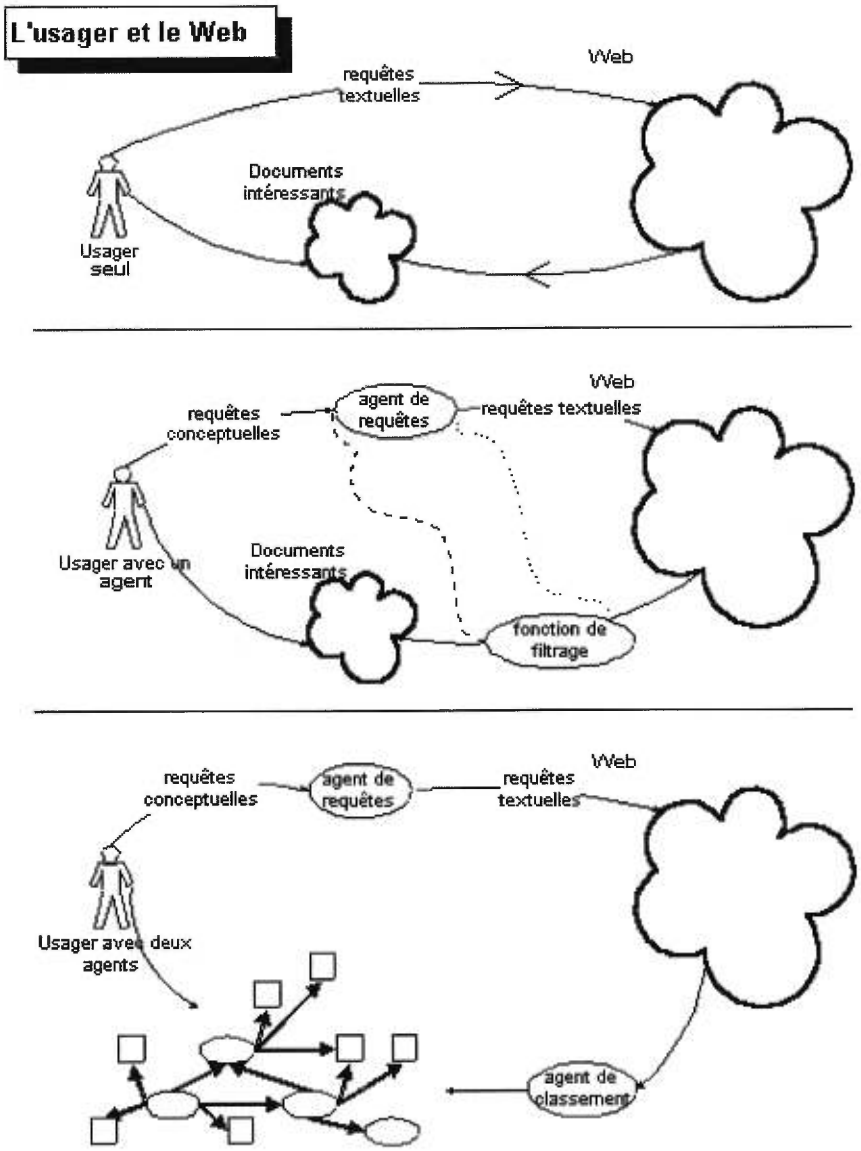

Figure 1-1: L'usager et le Web

Nous décrirons dans ce mémoire la conception et la réalisation d'un prototype pour un système d'enrichissement semi-automatique d'un curriculum par des recherches sur Internet.

1.2 Systèmes tutoriels et gestion d'information

Si l'on réexamine le problème de découverte et de gestion de ressources, cette fois dans le contexte des systèmes tutoriels, on constate encore une carence d'outils.

D'une part, il y a des outils qui peuvent servir à organiser, enchaîner et exploiter des ressources déjà découvertes, en partant de modèles d'intervention pédagogique comme celui de Gagné et en produisant au bout du compte des objets multimedia sophistiqués pouvant servir de tutoriels; on pense à ID2 [Merr92]. D'autre part, il y a des environnements de génie pédagogique et de gestion de curriculum comportant certaines des facettes requises d'un environnement de développement de système tutoriel intelligent; on pense ici à CREAM [Nkam96] et à AGD [Paqu96]. Finalement, on trouve tous les travaux sur les cartes cognitives, en particulier les recherches menées par B. Gaines et son équipe avec leurs KMaps, i.e., Knowledge Maps, ou cartes des connaissances [Gain95] ou encore l'environnement de construction et d'organisation de collections de ressources pédagogiques développé par l'équipe de J. Greer, le MicroWeb Toolkit [ThoJ95]. Aucun de ces outils ne semble supporter la découverte des ressources sur l'Internet. Leur intérêt réside ailleurs, dans le développement de cartes cognitives par exemple, ou dans le support d'un modèle étudiant ou encore dans la production d'objets hypermedia avec une certaine articulation d'un modèle conceptuel faible d'un domaine.

De tous ces travaux, c'est l'approche des cartes cognitives qui s'approche le plus des problèmes qui nous préoccupent: les ingrédients sont les mêmes, des structures de connaissances et un environnement de découverte intégré à l'Internet; mais les travaux de Gaines semblent porter plus sur des objets intelligents qui soient interprétables via des extensions au navigateur Netscape. *On pourrait dire que Gaines cherche à intégrer des structures de connaissances dans l'Internet alors que nous cherchons plus simplement à intégrer l'Internet dans des structures de connaissances.*

1.3 Aperçu des résultats originaux

Nous n'avons développé aucun nouvel algorithme pour résoudre le problème d'enrichissement automatique d'une structure de concepts. L'intérêt du prototype réside plutôt dans son exploitation de techniques classiques et modernes pour résoudre de façon satisfaisante un problème de gestion d'information.

Par exemple, l'idée de classer un nouveau document dans notre collection en le considérant comme une requête faite auprès de la collection et en sélectionnant la grappe de documents la plus pertinente, bien que conçue indépendamment, a été retrouvée par la suite dans [Thom95]. Et puis, il existe déjà quelques outils interactifs d'assistance au furetage mais ils s'attaquent plutôt à déterminer si un document est intéressant pour l'usager qu'à "comprendre" où situer un document dans son univers conceptuel; Letizia, par exemple, suivra les liens intéressants issus de la page courante, sans procéder à une exploration plus large [Lieb97].

L'idée de programmer et d'émettre des requêtes vers des engins de RI pour trouver des documents intéressants n'est pas neuve non plus, à preuve le support des engins de RI pour le standard CGI (Common Gateway Interface) ainsi que le support direct de langages comme Java et Perl pour ce genre d'opération. Finalement, les mécanismes de mise en page automatique du graphe de concepts et de documents ne sont pas neufs: ils ont été adaptés de classes Java qui reprennent des travaux désormais classiques sur les modèles à base de ressorts (voir chapitre II).

L'originalité de notre mémoire réside plutôt dans la façon dont nous avons intégré quelques techniques éprouvées pour former un environnement minimaliste et très souple pouvant supporter la tâche d'un concepteur pédagogique, d'un étudiant ou d'un travailleur intellectuel quelconque qui cherche et recherche des documents pertinents.

1.4 Plan des chapitres

Les chapitres s'organisent comme suit: le chapitre II commence par un rappel des principales notions et techniques en recherche d'information; ces techniques formeront l'essentiel des outils utilisés par le prototype; le chapitre se poursuit par un tour d'horizon des architectures ou prototypes s'attaquant à l'une ou l'autre des facettes du problème de filtrage d'information sur Internet, notamment cueillette, sélection et affichage. Au chapitre III, nous circonscrivons un problème spécifique en gestion d'information - celui de l'enrichissement d'un curriculum - et nous décrivons notre approche de solution. Le chapitre IV détaille la modélisation conceptuelle d'un prototype s'attaquant à ce problème. Le chapitre V décrit la modélisation physique, au niveau des classes Java et de leurs interactions. Le chapitre VI rend compte des expériences menées pour évaluer le prototype. Enfin, au chapitre VII, après avoir discuté les lacunes et exposé les améliorations souhaitées le long de divers axes, nous prenons le recul nécessaire pour situer le prototype dans le paysage conceptuel de la gestion d'information et éventuellement de la gestion des connaissances.

Chapitre II: État de l'art en gestion d'information

Dans la littérature de la gestion d'information, il est souvent question de *filtrage*: un système de filtrage d'information est un SRI qui construit un modèle de l'usager - ou de sa communauté - et qui est donc mieux à même de cibler ses besoins. Dans leur mémoire sur le filtrage d'information textuelle, Oard et Marchinioni [Oard96] replacent d'abord le problème du filtrage dans *le cadre global de la médiation automatique de l'accès à l'information présente sur un réseau.* Ils font état de trois étapes essentielles, notamment cueillette, sélection et présentation. La cueillette représente l'étape pendant laquelle le système et l'usager transforment un besoin d'information relativement spécifique en une liste de sources d'information, par exemple des documents, susceptibles de satisfaire ce besoin. Pendant l'étape de sélection, la liste des ressources ainsi obtenues est analysée et raffinée pour ne retenir que les documents les plus importants dans le contexte. Finalement, le système présente ses résultats à l'usager. La figure 2-1 illustre ces trois étapes avec leurs entrées et leurs sorties.

Ces trois facettes de cueillette, sélection et présentation sont déjà représentées à des degrés divers par des articles, prototypes et algorithmes documentés. Nous les illustrerons au moyen d'exemples qui nous semblent offrir un bon point de départ pour la solution de problèmes somme toute assez difficiles. Nous commencerons ce chapitre par une introduction aux techniques de repérage de l'information; ces techniques alimentent les volets cueillette et sélection de tous les systèmes de gestion d'information. Nous parlerons ensuite de prototypes comme MIDS (2.2.1) et Syskill & Webert (2.2.2), qui sont particulièrement remarquables pour la cueillette et la sélection. Pour donner un aperçu des travaux en présentation d'information, nous discuterons GSA (2.3.1) et les modèles à base de ressorts (2.3.2). Finalement, puisque l'apprentissage améliore la performance des systèmes de gestion d'information, nous décrirons NewsWeeder. Plutôt que d'énumérer tous les aspects de chaque architecture, nous retenons pour chaque problème des solutions susceptibles de nous aider à résoudre notre problème d'enrichissement d'un curriculum.

2.1 Techniques de repérage d'information

Le présent mémoire décrit la construction d'un outil d'enrichissement de curriculum par recherche sur Internet. Il faut modéliser un besoin d'information, des documents, des collections et traiter tous ces objets de manière à pouvoir aller chercher des documents pertinents et leur assigner un point d'attache

dans une structure de concepts définie par l'usager. Or il existe une discipline dont la préoccupation centrale est la suivante: Est-ce que le document D est pertinent au besoin d'information de l'usager? Cette discipline, c'est la RI (Recherche d'information). Nous allons donc, dans cette section, présenter les notions et résultats du domaine de la RI qui vont nous être utiles dans la solution de notre problème.

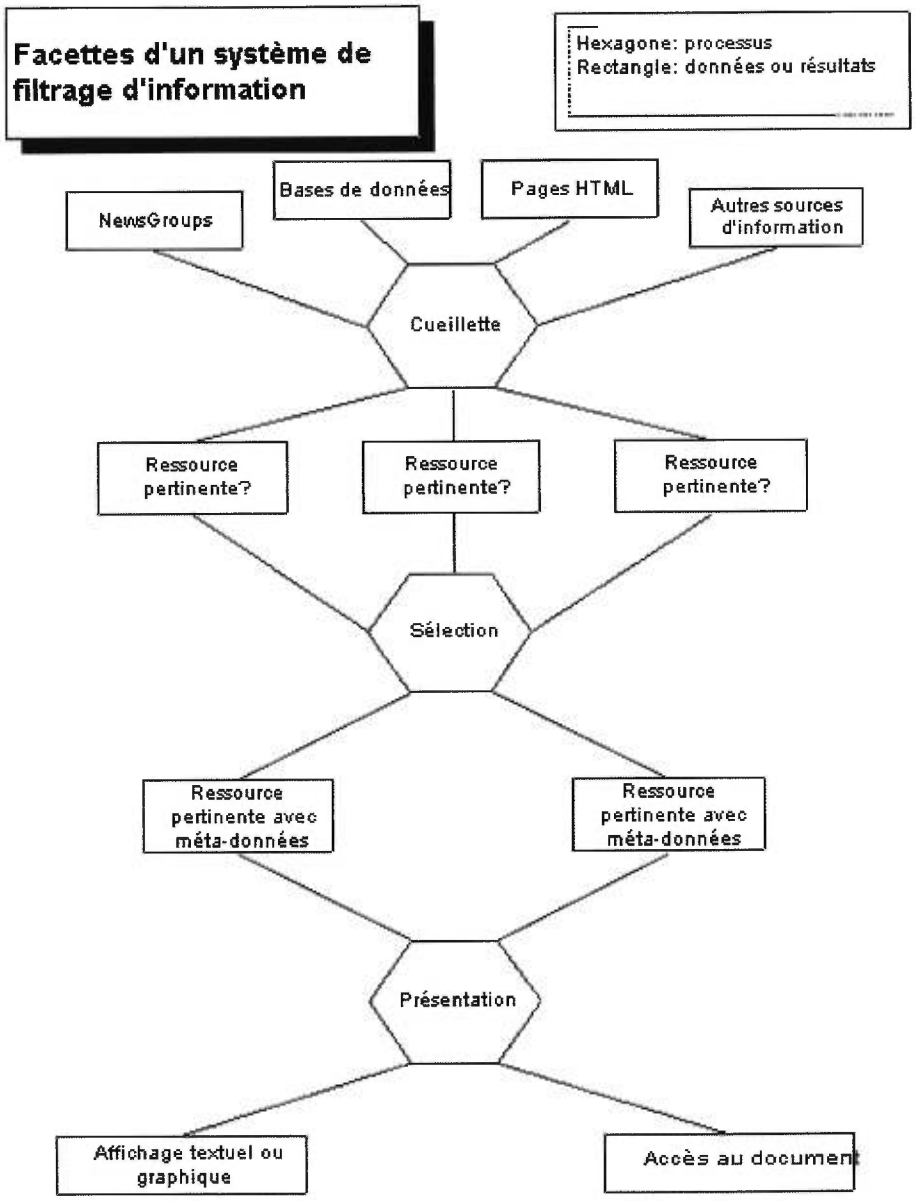

Figure 2-1: Facettes d'un système de filtrage d'information

On commencera par un portrait général de la problématique en RI et du modèle vectoriel en particulier. Puis on décrira plus en détail la construction des modèles. Cette construction passe par la sélection des termes et par leur pondération, c'est-à-dire, par l'indexation. Si le système construit un modèle de

l'usager, la qualité des réponses pourra être améliorée: on introduira ces raffinements en parlant de la rétroaction de pertinence. On évoquera ensuite le rôle que peuvent ou doivent jouer les connaissances de type linguistique dans les systèmes de repérage d'information. Enfin on abordera les questions d'évaluation en RI.

2.1.1 Modèles de données et modèles de traitement

Différentes approches sont disponibles en RI pour la représentation des documents et des requêtes ainsi que pour le calcul de pertinence. À côté des modèles classiques, non-intelligents, on compte des approches plus modernes incorporant la logique et le traitement de langue naturelle (TLN). Nous avons choisi un modèle classique, le modèle vectoriel, pour des raisons de simplicité et parce que la rétroaction de pertinence s'y implante très naturellement. Nous ne proposons pas dans ce chapitre une introduction systématique à la recherche d'information: nous nous concentrerons sur le modèle vectoriel. Quelques bonnes introductions au domaine sont mentionnées dans la bibliographie, en particulier [Frak92], [Rijs79] et [Salt83]. Par contre, nous avons cru bon de situer cette problématique de façon générale au moyen d'une carte conceptuelle, trouvée ci-dessous.

Cette carte conceptuelle met en évidence l'importance de la modélisation en RI. *Le besoin d'information de l'usager et les collections de documents sont des objets incommensurables: on s'attaque au problème en construisant des représentations du besoin d'information et des représentations des documents;* le problème s'en trouve simplifié puisqu'il suffira ensuite de faire un calcul de "comparaison" entre les deux types d'objets au moyen des représentations choisies. La représentation des documents et du besoin d'information peut être plus ou moins riche: plus cette représentation est riche, plus elle supportera un calcul sophistiqué. Les modèles classiques offrent une représentation pauvre, basée sur l'indexation.

Les premiers systèmes de RI étaient de type booléen: ils détectaient la présence ou l'absence d'un terme et fournissaient une réponse binaire à la question de pertinence. Outre le fait que la pertinence se laisse mal décider sans nuances, ces systèmes retournaient une liste désordonnée de documents. Avec la prochaine étape, le modèle vectoriel, on voulait éviter de trancher par un oui ou par un non en donnant une réponse de type quantitatif: le document D est plus ou moins pertinent pour la requête Q. Cette pertinence, un nombre réel, est obtenue par un calcul de ressemblance entre Q et D. Ainsi, pour une collection donnée, on obtiendra les documents les plus pertinents en effectuant un calcul de ressemblance entre la requête et chacun des documents à son tour, puis en triant la liste par ordre de ressemblance.

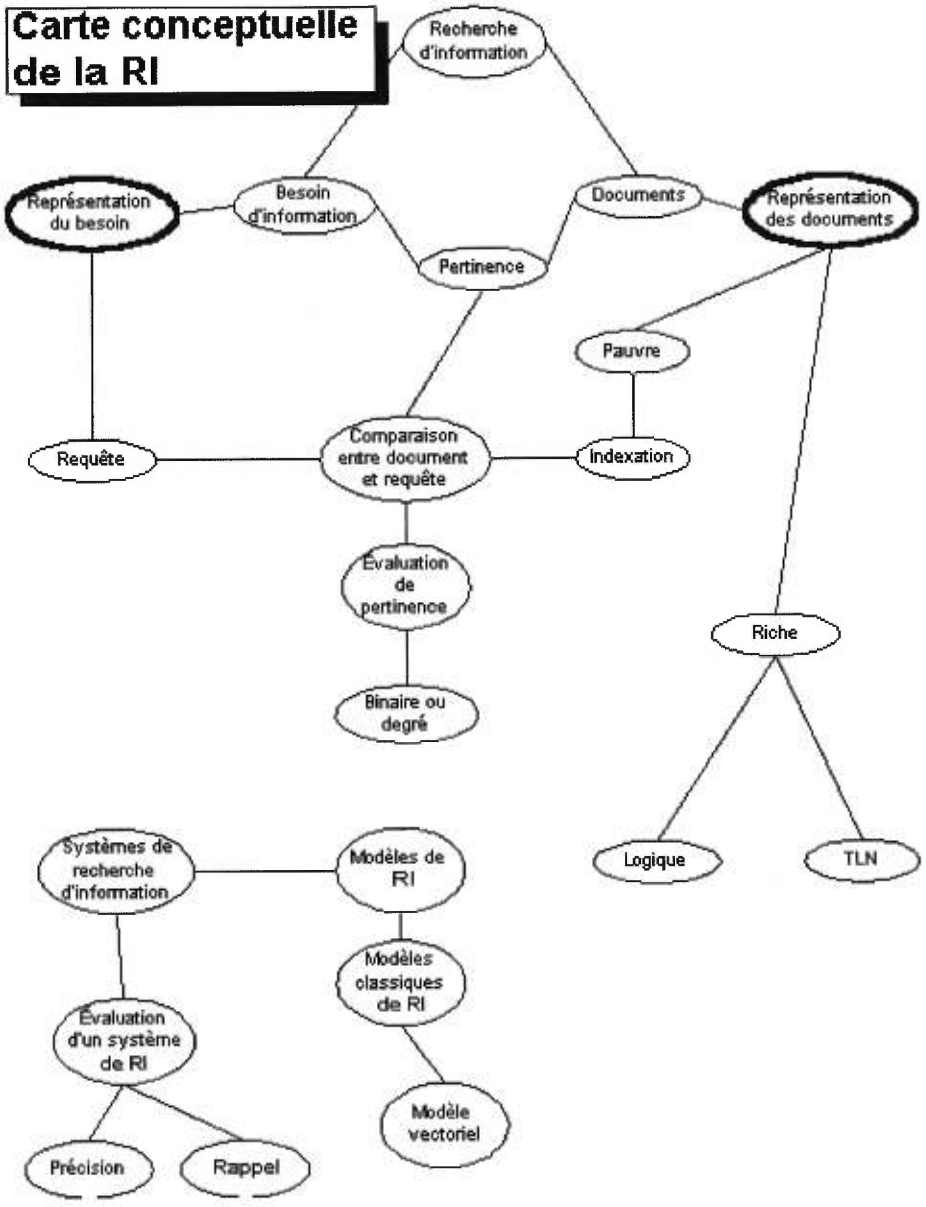

Figure 2-2: Carte conceptuelle de la Recherche d'information

Dans le modèle vectoriel de RI, pour calculer la ressemblance, on représente les documents et la requête comme des vecteurs et on mesure une distance entre ces vecteurs. Plusieurs décisions doivent être prises: quels sont les vecteurs de base, comment calculer les poids et comment calculer la distance?

2.1.2 Le modèle vectoriel en RI

Puisqu'on a décidé de transformer le problème en un calcul de ressemblance dans un espace vectoriel, chaque document et chaque requête seront représentés par une expression de la forme: $< w_1X_1, w_2X_2, \ldots, w_nX_n>$ où Xi est un terme (index) et wi son poids. L'indexation d'un document, c'est justement une façon d'en arriver à cette représentation. Un algorithme d'indexation va devoir passer par les étapes suivantes:

- un analyseur lexical produit une liste de termes candidats pour les documents de la collection
- des statistiques sur les termes sont produites et analysées
- un ensemble de termes d'indexation est sélectionné
- les poids des ces termes sont assignés

Le résultat est l'espace vectoriel représentant la collection. Les X_i s'appellent *termes d'indexation.* Comment va-t-on choisir les termes susceptibles de représenter le mieux possible le contenu d'un document?

Indexation - Sélection et pondération des termes

Il serait bon de préciser quel type d'objets nous cherchons à sélectionner, c'est-à-dire de quel type sont les X_i dans la représentation du document comme $< w_1X_1, \ldots w_nX_n >$. Ils peuvent être dérivés du texte ou obtenus d'une source indépendante. Le vocabulaire peut être constitué de mots isolés, d'expressions ou de lemmes (représentations tronquées). Dans le cas d'une indexation automatique, les critères de sélection les plus importants sont la fréquence et la discrimination.

La littérature des SRI contient plusieurs études sur les diverses métriques et leurs mérites respectifs; nous nous inspirerons surtout de [Zobe98]. *La ressemblance $S_{q,d}$ entre un document et une requête est généralement basée sur un jugement de l'importance d'un terme pour le document, $w_{d,t}$; l'importance du terme pour la requête, $w_{q,t}$; le poids du document, W_d et le poids de la requête, W_q.* La ressemblance peut prendre la forme d'un produit scalaire du vecteur requête avec le vecteur document, du cosinus de l'angle entre le document et la requête, ou encore il y a quelques mesures probabilistes. Comme nous avons choisi d'utiliser un cosinus dans le prototype, nous en donnerons ici la formule:

$S_{q,d} = \Sigma(w_{q,t} * w_{d,t}) / \sqrt{((\Sigma w_{q,t}^2) * (\Sigma w_{d,t}^2))}$ où la somme est prise sur tous les termes

Équation 1: Calcul de ressemblance avec la formule du cosinus

La notion de poids d'un terme correspond à l'intuition que les termes qui apparaissent dans plusieurs des documents de la collection sont moins importants - dans une optique de calcul de ressemblance - que ceux qui n'apparaissent que dans quelques-uns des documents. Le terme t se verra donc assigner un poids w_t, souvent appelé IDF pour "Inverse Document Frequency", qui reflète son degré de spécificité. En pratique, le calcul de poids se fait selon une approche binaire, logarithmique, hyperbolique ou entropique. Nous avons retenu l'approche hyperbolique:

$(4.2) w_t = \log_e(1 + N / f_t)$, où:

- N est le nombre de documents
- f_t est le nombre de documents contenant le terme t

Équation 2: Calcul de poids avec l'approche hyperbolique

Une fois déterminés les termes d'indexation et leur pondération, le travail se réduit à une arithmétique de produits scalaires. Cette approche simple au calcul des représentations et de la pertinence a fait ses preuves. Cependant, elle a ses limites. Ces limites pourraient-elles être reculées en situant le SRI dans une boucle, en lui offrant une rétroaction?

2.1.3 Rétroaction de pertinence

Les systèmes de RI sont limités dans leur performance par la nature textuelle de leurs mécanismes de recherche: a priori, si la requête contient le terme T et que le document, bien que pertinent, est indexé au moyen du terme T', le document ne sera pas repéré. Le repérage est généralement littéral plutôt que sémantique. Toute faute d'orthographe aura donc un impact négatif sur le taux de rappel de la requête, c'est-à-dire sur la proportion de documents pertinents qui seront effectivement jugés tels par l'engin. Pour pallier à ces problèmes, *on voit de plus en plus de systèmes qui offrent une rétroaction de pertinence. La rétroaction de pertinence, telle qu'utilisée en RI, est l'une des formes les plus simples d'apprentissage que l'on puisse imaginer. L'usager peut indiquer au système lesquels parmi les documents repérés sont pertinents, lesquels le sont moins ou pas du tout. Le système procède alors à une réassignation des poids de termes ainsi qu'à une reformulation de la requête, le tout pour tenter d'obtenir de meilleurs résultats.* Le changement de poids permet aux documents pertinents déjà repérés d'avancer vers la tête de la liste.

Les poids dans les vecteurs sont ajustés de façon à rapprocher le vecteur requête des vecteurs documents pertinents et à l'écarter des documents non-pertinents. La reformulation de la requête permet d'aller chercher des documents indexés selon un vocabulaire différent du vocabulaire natif de l'usager. Il est à noter que dans un contexte classique de RI, les changements ci-dessus sont éphémères puisque l'engin ne possède pas de connaissances individualisées sur ses usagers. Par contre, dans un contexte de filtrage d'information, le système peut exploiter et enrichir sa connaissance de l'usager au moyen de ce que l'on appelle souvent dans la littérature un profil (voir 2.2.2, Syskill & Webert).

L'essentiel du traitement en RI consiste donc en la construction des modèles de documents et de collection, puis en la comparaison du modèle de requête avec ces premiers modèles: la comparaison fournit des nombres et le jugement de pertinence est apporté en considérant ces nombres. Les résultats sont satisfaisants, pourtant on s'attendrait à ce que des traitements de langue naturelle soient impliqués dans un calcul de pertinence.

2.1.4 Traitement langue naturelle

Nous avons jusqu'à présent adopté un point de vue résolument statistique: l'essence de ces méthodes consiste en l'attribution de hauts scores aux termes qui semblent de bons indicateurs de contenu, ce qui va leur donner une plus grande influence sur le rang des documents repérés. *Or comme les opérations d'indexation et de repérage se déroulent de plus en plus dans un contexte de langue naturelle plutôt que dans des systèmes de descripteurs rigides, on pourrait s'attendre à ce que le traitement de langue naturelle (TLN) joue un rôle plus important.* On peut caractériser ces traitements de façon superficielle en disant que le traitement lexical s'attaque à la segmentation du texte en unités atomiques, le traitement syntaxique aux relations entre ces unités telles qu'exprimées dans la langue naturelle en question et finalement le traitement sémantique étudie les rapports entre les groupements textuels et les concepts dans ce qu'on appelle généralement la réalité. Les apports possibles de ces traitements en RI se situent au niveau du choix des unités, de la possibilité de méta-données et enfin de mécanismes plus riches.

Choix des unités, méta-données et mécanismes de comparaison

L'un des apports du TLN pourrait être au niveau du choix des mots et expressions qui constituent la représentation des documents. Par exemple, des connaissances sur les parties du discours pourraient être

exploitées au cours de la reconnaissance d'expressions: au lieu de ne considérer que des juxtapositions de termes, on pourrait exiger que ces dernières constituent en fait un syntagme nominal, par exemple un nom et un autre qui le détermine (Voir 2.2.1, MIDS). D'autre part, le problème de la variabilité d'expression des concepts complique beaucoup le repérage. L'approche traditionnelle normalise les termes, par exemple une normalisation morphologique comme la troncature ou une normalisation sémantique ou statistique à travers l'usage de classes ou de grappes. Un autre apport du TLN pourrait résider dans une structure de connaissances attachée au document et contenant l'information de relations paradigmatiques entre termes: ceci permettrait des substitutions contrôlées[Lewi96]. On aimerait permettre à des termes non-identiques de s'évoquer l'un l'autre.

Enfin les mécanismes de recherche et de comparaison auraient beaucoup à gagner en incorporant des connaissances et des algorithmes en langue naturelle. Un modèle plus profond permettrait d'aller au-delà des comparaisons exactes de chaînes de caractères ou de lemmes. Si l'on combine ceci avec l'apport d'une structure classificatoire, on obtient un système qui traite les mots comme des pointeurs vers des concepts plutôt que des simples chaînes de caractères.

L'intégration de traitements linguistiques plus sophistiqués dans les systèmes de RI existants n'apparaît pas très prometteuse à l'heure actuelle; nous ne nous en servirons pas dans cette recherche. Par contre, l'exploitation de ressources comme des thesauri est une possibilité attrayante. Quelques ressources de ce type sont maintenant disponibles dans le domaine public et il est plus facile de les utiliser que d'essayer d'intégrer des modules TLN dans un système. Cependant il est permis de croire que les recherches en extraction d'information, un domaine en effervescence à l'heure actuelle, vont éventuellement déboucher sur des architectures et des algorithmes que l'on pourra intégrer en RI.

La RI nous fournit donc plusieurs façons de construire automatiquement des représentations des documents pour évaluer ensuite la pertinence d'un document pour une requête; parmi ces approches, nous avons choisi le modèle vectoriel et une ressemblance basée sur un cosinus. Mais la RI nous fournit-elle aussi une façon d'évaluer la qualité des résultats? La RI semble dominée par les approches statistiques; on s'attend donc à ce que l'évaluation des SRI soit une sous-branche particulièrement bien développée de cette discipline.

2.1.5 Les métriques en RI

L'évaluation est un aspect difficile des systèmes de repérage d'information. Les publications sur la RI offrent rarement un traitement approprié à la question de l'évaluation. Le livre de Salton et McGill

[Salt83] ainsi qu'un livre de Rijsbergen [Rijs79] font exception. Dans [Rijs79] l'auteur tente d'établir et d'enseigner un cadre méthodologique. Dans un chapitre consacré aux questions d'évaluation, il commence par poser les questions: *Pourquoi évaluer? Quoi évaluer? Comment évaluer?* Et il fait état des réponses apportées par Cyril Cleverdon en 1966.

Cleverdon fut l'un des premiers chercheurs à tenter d'établir la RI sur des bases expérimentales plus solides. À la question *Quoi évaluer*, il offre les réponses suivantes:

- la couverture d'une collection
- le temps-réponse pour une requête
- la qualité de la présentation des résultats
- l'effort fourni par l'usager pour obtenir des réponses à ses questions
- le rappel du système
- la précision du système

Bien qu'elles soient d'une grande importance, les quatre premières quantités ont moins retenu l'attention: le rappel et la précision sont devenus les deux mesures classiques de performance associées aux SRI. *Le rappel désigne la fraction des documents repérés parmi les documents pertinents accessibles au SRI. La précision désigne la fraction des documents pertinents parmi les documents repérés par le SRI. Ces deux mesures croissent en sens inverse pour un système*: plus on récupère de documents, plus on risque d'en récupérer qui n'étaient pas vraiment pertinents à la requête. Cleverdon a aussi construit les premières collections de test, indispensables pour rendre les résultats des évaluations répétables.

Limitations de la précision et du rappel

Les recherches classiques en RI portaient sur des collections statiques: l'univers des documents était clos, ainsi la notion de rappel était bien définie. Elle l'est beaucoup moins de nos jours, quand une large proportion des requêtes faites chaque jour à des SRI visent en fait l'univers mouvant du Web: les serveurs deviennent accessibles ou inaccessibles, les documents changent de nom, de point d'attache, de contenu. Comment évaluer le rappel dans ces conditions? Un autre problème a trait à la nature des documents et des interactions qu'ils supportent: avec l'hypertexte et l'hypermedia, la question des frontières du document devient très floue: que cherche l'usager, un passage, la racine d'une grappe de pages Web, une table des matières? Chercher et trouver prennent un nouveau sens. De plus, l'aspect temporel d'un document peut être partie intégrante de sa pertinence ou de sa non-pertinence: par exemple, trouver tous

les brevets autour de tel mécanisme émis après telle date. On se rapproche de bases de données textuelles ou même de bases de connaissances. On se rapproche des systèmes d'extraction d'information.

Néanmoins la principale objection à ces mesures est qu'elles reposent sur une notion de pertinence fragile et bien mal définie. Dans [Mizz98], quatre dimensions de la pertinence sont présentées: les sources d'information, la représentation du besoin de l'usager, le temps et les composantes. Les cadres d'évaluation comme TREC font l'hypothèse que la pertinence est une propriété objective de la relation entre un document et un concept. Ne faut-il pas plutôt envisager la pertinence comme une relation entre un usager, un concept et un document? De plus, il est à noter que la qualité d'une référence est généralement ignorée dans l'évaluation; toutefois, les SRI sur Internet commencent à incorporer des mesures simples, sous la forme du nombre de liens conduisant à une page donnée. ([Li98], [Thom98]).

Nous avons décrit dans cette section les méthodes de recherche d'information en nous concentrant sur l'approche vectorielle, retenue pour sa simplicité et son efficacité. Nous avons évoqué la place des traitements de langue naturelle pour nous rendre compte que cette place n'est pas encore très large à l'heure actuelle en RI. Enfin, nous avons présenté les principales métriques d'évaluation en RI, notamment rappel et précision. Au terme de ce survol, nous avons les outils nécessaires pour nous attaquer au problème de la gestion d'information sur Internet. Le peu d'impact du TLN en RI, bien que déplorable, nous permet de croire qu'il est tout de même possible d'obtenir des résultats sans un arsenal d'outils linguistiques. Toutefois, avant d'entreprendre la conception de nos outils de requête et de classement, nous avons fait un tour d'horizon des systèmes qui s'attaquaient à ce problème ou à des problèmes connexes; nous décrirons maintenant les résultats de cette exploration.

2.2 Systèmes de cueillette et de sélection d'information

Pour illustrer les possibilités de cueillette et de sélection, nous avons retenu MIDS et Syskill & Webert. Ces produits sont diamétralement opposés en terme de complexité et d'intégration à l'environnement de l'usager. Alors que MIDS repose sur une architecture Harvest et intègre toutes sortes de traitements pour fournir à l'usager des outils puissants d'accès à une information répartie, Syskill est un petit logiciel qui exploite de vastes connaissances en apprentissage pour enrichir l'expérience de navigation Netscape d'un usager quelconque. Syskill a peu à offrir côté présentation des résultats, MIDS un peu plus, mais nous négligerons la facette présentation pour y revenir à la section suivante.

2.2.1 Cueillette et organisation d'information - MIDS

MIDS, le MITRE Information Discovery System, est un environnement intégré de découverte, classification et navigation d'information développé par les chercheurs de l'institut MITRE [Helm95]. C'est un projet ambitieux qui vise non seulement la cueillette et la catégorisation de l'information, mais s'attaque aussi à la structuration des ressources repérées. Ceci passe par la production de méta-données à partir d'une population hétérogène d'objets. MIDS offre des pages blanches et des pages jaunes, c'est-à-dire que les items repérés sont accessibles par nom ou par sujet. MIDS s'articule sur deux composantes principales: l'organisateur d'information et le courtier en information.

Architecture Harvest

La plate-forme logicielle choisie pour l'infrastructure de ce système est Harvest [Bowm94] développé par le Groupe de recherche sur la découverte des ressources de l'Internet Research Task Force (IRTF). *Les recherches qui ont mené vers Harvest se préoccupaient essentiellement du fait que l'explosion de l'Internet rendait peu à peu désuets des outils de découverte qui avaient jusque-là joué un rôle crucial.* Le but de Harvest est de fournir un cadre d'une grande souplesse pour la cueillette et la gestion de l'information sur l'Internet. Harvest est une architecture à quatre parties: les Gatherers, les Brokers, les Réplicateurs et les Caches d'objets.

MIDS utilise des Gatherers et des Brokers Harvest.

Composantes Harvest de MIDS

Un Gatherer est une composante qui recueille des méta-données pour l'indexation auprès de divers fournisseurs d'information, c'est-à-dire, dans la première version, des serveurs de fichiers MITRE. Les Gatherers connaissent divers types d'objets, comme les fichiers HTML, texte, PostScript et ils sont en mesure d'analyser ces fichiers pour en extraire les méta-données requises. Le format adopté pour MIDS s'appelle SOIF, pour Summary Object Interchange Format; il est basé sur une intégration du format des archives FTP (IAFA) et du format BibTex. Les entrées en SOIF sont de type <attribut><valeur>.

Un Broker quant à lui permet de fouiller l'information acquise par les Gatherers ou par d'autres Brokers. Il se spécialise dans certains types d'information. Le Broker Information Service présente à l'usager un espace d'information unifié, construit à partir des espaces d'information de divers Brokers.

Composantes originales

Le GDS (Gatherer Dissemination Service) est une composante propre à MIDS. Périodiquement, le GDS recueille les données SOIF auprès de plusieurs Gatherers. Tous les documents repérés continuent de résider chez leur fournisseur respectif: ce sont seulement les métadonnées qui sont traitées et entreposées dans le MIDS. Le GDS possède un moteur de classification qui travaille sur les méta-données d'un document plutôt que sur leur texte. GDS offre une classification de type externe, basée sur une taxonomie, des règles et des descripteurs fournis par un cogniticien. Ce moteur est basé sur l'engin de filtrage SIFT [YanG95]. GDS offre aussi une classification de type interne au moyen d'outils de regroupement automatique; ce deuxième type d'outil est utilisé pour offrir, à l'intérieur de la taxonomie externe, une granularité plus fine qu'il serait difficile de calculer autrement. GDS distribue les méta-données chez divers Brokers.

Une des forces de MIDS est la palette d'outils fournis à l'usager pour traiter les documents repérés, en particulier, la création de sommaires. Ces outils ont comme point de départ un identificateur des parties du discours (POS Tagger) basé sur les travaux de Brill mais modifié par MIDS; ici, la prémisse est que seuls comptent noms et verbes; on produit pour chaque document un vecteur ne contenant que ces derniers. Ce vecteur est ensuite utilisé à des fins de construction de sommaires ou de regroupement de documents.

Le principal intérêt de MIDS réside dans la façon dont on a tiré parti de diverses techniques existantes pour monter un cadre de travail puissant et extensible. L'intégration au niveau de l'interface d'un cadre de classification des documents dans une taxonomie existante avec une répartition subséquente des documents en grappes dynamiques devrait faciliter grandement la maîtrise d'un espace d'information. De plus, la bande passante est gérée de façon économique: seules les méta-informations sont transportées. *Par contre, pour ce qui est du problème qui nous intéresse, on note que MIDS est plutôt un outil de catalogage qu'un outil de recherche sélective d'information.*

2.2.2 Formulation des requêtes - Syskill & Webert

Le prototype Syskill & Webert par Pazzani, Muramatsu et Billsus, était disponible en 1996-1997 sur le Web [Pazz96]. Nous commencerons par donner un aperçu du système complet, puis nous nous concentrerons sur le volet que nous avons retenu, en l'occurrence la formulation automatique de requêtes pour satisfaire un besoin d'information.

Construction et exploitation des profils

Le système est construit autour de profils d'usager, à raison d'un profil par centre d'intérêt, par usager. Dans sa phase d'apprentissage, le système demande à l'usager d'évaluer une centaine d'URL contenus dans une page portant sur un sujet donné. L'usager indique son intérêt, indifférence ou rejet de chaque URL. Syskill & Webert recueille les documents HTML et leur évaluation et les transforme en des vecteurs de paires <attribut, valeur> qu'il soumet ensuite à un programme d'apprentissage. Quand un usager évalue une page, le document source HTML est copié à un fichier local et une entrée est faite pour mémoriser le classement: cette entrée contient la classification - chaud, tiède ou froid - l'URL, le nom du fichier local, la date à laquelle le fichier a été copié ainsi que le titre de la page.

Le résultat - un profil de l'usager - est exploité de deux façons: pour construire des requêtes Lycos et pour évaluer si les URL contenus dans une "page courante" sont prometteurs. La deuxième fonctionnalité est une aide à la navigation et tombe dans la catégorie agent d'interface décrite par Lieberman dans [Lieb95] Pendant que l'usager navigue, le système travaille dans les coulisses pour suivre les liens qui émanent de la page et les classer en trois catégories d'intérêt pour l'usager, nommées chaud, tiède et froid. Le système présente ses résultats dans une nouvelle fenêtre du fureteur Netscape: les liens y sont précédés d'une icône appropriée.

Comme le rapportent les auteurs eux-mêmes, les suggestions de navigation sont surtout utiles si l'usager possède déjà un ensemble de liens qui couvre bien son sujet. C'est un type de considération qui interviendra aussi dans le prototype que nous avons construit. *Remarquons de plus que le contexte, le niveau de granularité, est ici un sujet, un concept tout entier, plutôt qu'une structure de concepts: la classification d'un document est ou bien très pertinent, ou bien un peu, ou bien pas du tout pour le sujet entier. Cette granularité n'est pas assez fine pour le problème qui nous intéresse.*

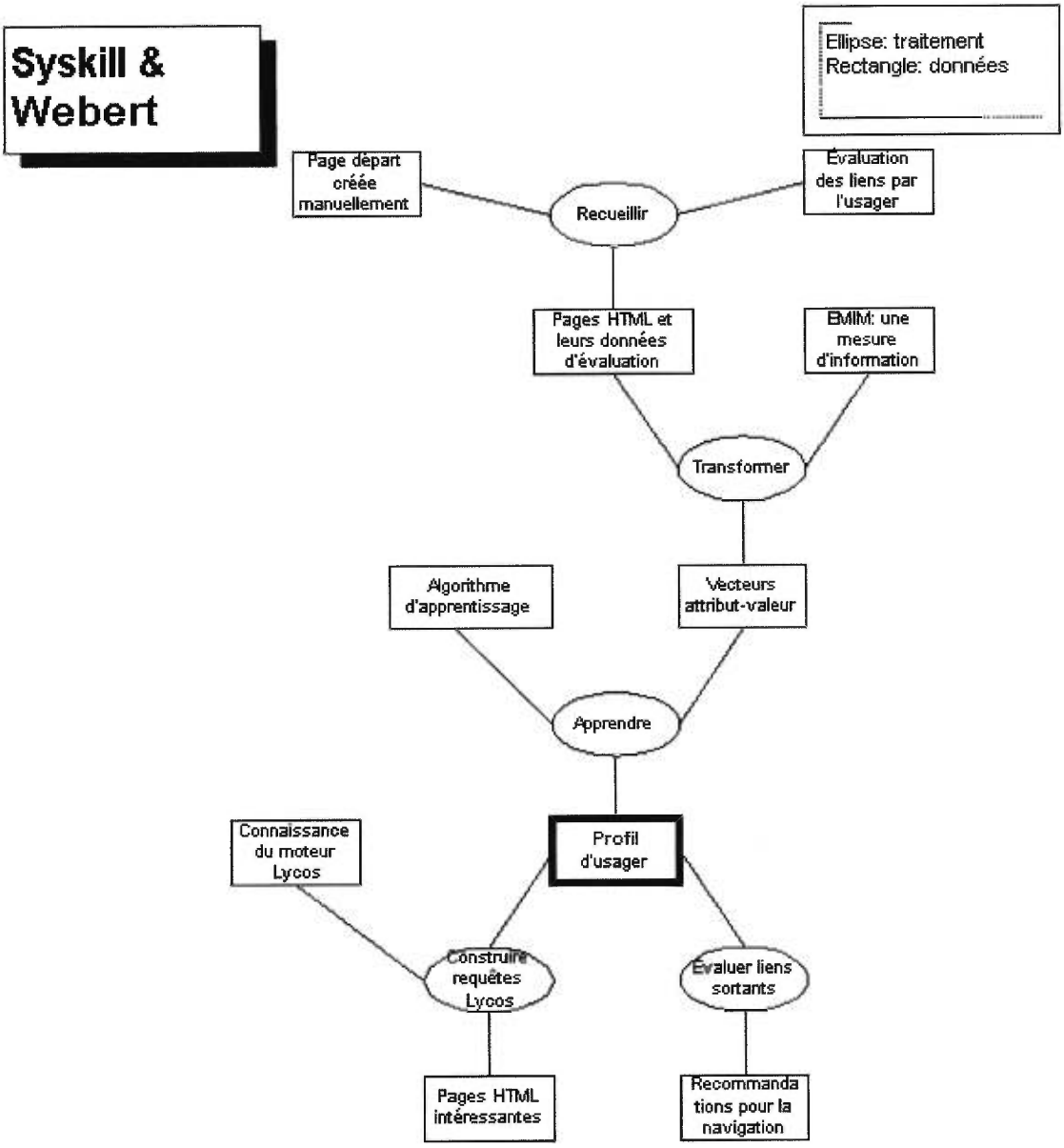

Figure 2-3: Modèle du prototype Syskill & Webert

Formulation automatique des requêtes

La facette de ce système qui nous intéresse ici est celle de la formulation automatique des requêtes. Notons que l'usager peut avoir plusieurs profils, un par domaine d'intérêt. Le système exploite un profil pour construire des requêtes qui doivent repérer des documents intéressants pour l'usager. L'approche choisie est la suivante: puisque la performance de Lycos est faible pour les requêtes trop longues, les

requêtes sont formées de quatorze mots: 7 termes génériques pour définir le domaine, i.e., les ressemblances entre les documents de départ portant sur un sujet donné, intéressants ou non; 7 mots pour cerner les différences entre les documents intéressants et les autres. Parmi ces 14 mots, sept seront des termes spécifiques glanés dans les documents déjà évalués: ce sont les sept mots avec la plus haute valeur de discrimination pour les pages intéressantes. Un exemple de requête pour un usager qui s'intéresse aux sciences du vivant: mots spécifiques - *grants, control, WUSTL, data, genome, CDC, infectious*; mots génériques - *university, research, pharmacy, health, journal, biology* et *medical*.

Les auteurs rapportent des expériences visant à améliorer la performance de leur prototype: ils ont fait des essais en retenant seulement les N termes les plus spécifiques, et d'autres en examinant seulement les M premiers termes d'un nouveau document, ceci pour diverses valeurs de M et de N. On pourrait tirer parti des ces optimisations. Somme toute, Syskill & Webert démontre la faisabilité de la production automatique et du lancement automatique de requêtes vers un SRI sur Internet. Par contre, comme nous l'avons déjà mentionné, le classement n'a pas la finesse requise pour notre application.

2.3 Systèmes de présentation d'information

L'une des facettes intéressantes en gestion de l'information est la possibilité ou la promesse de métaphores visuelles pour aider l'usager à comprendre de grandes quantités d'information. Nous nous intéresserons surtout au cas où il s'agit de s'orienter dans une collection de documents. *Généralement, quand on parle de visualisation, il peut s'agir de visualiser deux choses: les attributs d'une population ou encore les liens entre les différents membres de cette population. Dans le cas d'une collection de documents, les attributs sont des caractéristiques des documents individuels et les liens représentent une structure, implicite ou explicite, de relations entre documents.*

Ces caractéristiques et cette structure forment la matière brute fournie au module de visualisation. Les résultats obtenus en sortie s'articulent autour de deux décisions: la position d'un point-document dans l'univers visible et les propriétés de l'objet graphique qui va le représenter. Ce calcul de position pose un problème intéressant: il est rare que les membres de la population d'information soient situés naturellement dans un espace à deux ou trois dimensions: le navigateur aura donc des problèmes d'orientation. Nos cartes géographiques sont interprétées au moyen d'une traduction du mouvement ambulatoire corporel en un mouvement oculaire, mais comment créer une carte d'un espace virtuel?

Une solution qui s'impose de plus en plus pour traiter de grandes quantités d'information est une approche que l'on pourrait appeler relationnelle au sens où elle met l'accent sur les liens entre les objets plutôt que sur les objets eux-mêmes. Dans la littérature de l'hypertexte, cette situation est décrite par la métaphore du rhizôme, due à Deleuze et Guattari [Burn98]; ces sémioticiens se sont attaqués à la caractérisation du nouvel espace de communication post-moderne inauguré par l'hypertexte et ils ont dégagé certains de ses traits essentiels, comme l'impossibilité d'assigner à un document une position et un contenu fixes et le primat des liens entre les textes.

2.3.1 Structuration automatique de l'information - GSA

Le problème est donc le suivant: étant donné une grande quantité de documents, comment produire une représentation visuelle intelligible qui se prête à la manipulation directe et à la navigation? Une façon naturelle d'attaquer ce problème est de découpler la production de la structure et la production de la représentation visuelle. Dans le projet que nous avons retenu, on vise à produire des réseaux PathFinder incarnés dans un monde virtuel et la solution au problème de production de la structure passe par GSA (Generalized Similarity Analysis) [Chen98].

Réseaux PathFinder

GSA offre un cadre de travail pour extraire une structure d'un espace hypermedia. L'objectif est d'en arriver à un espace qui s'auto-organise. Pour chaque paire de documents, on va calculer une ressemblance qui sera un point dans l'ensemble des nombres réels. Chaque document étant représenté par un nombre naturel, on s'intéresse à la ressemblance entre les paires de documents, représentée par une fonction Ress() des paires de nombres naturels dans l'ensemble des nombres réels. À partir de cette fonction, on va construire un réseau de navigation. Mais au lieu de prendre comme point de départ pour le réseau toutes les relations ainsi construites (en utilisant un simple seuil pour alléger) on procède à une simplification qui ne retient que les relations les plus saillantes. Intuitivement, le raisonnement est le suivant: si le lien entre un document B et un document C **gagne** à être décomposé comme B ==> A ==> C, on élimine ce lien d'emblée: il sera retrouvé par composition des deux chemins. Reportons-nous à la figure 2-4. Puisque $2.0 > 0.5 + 0.5$, il est plus efficace de naviguer de B à C en passant par A qu'en employant le chemin direct entre B et C.

Cet algorithme n'est pas sans parenté avec les algorithmes de calcul du minimal spanning tree [Rijs79].

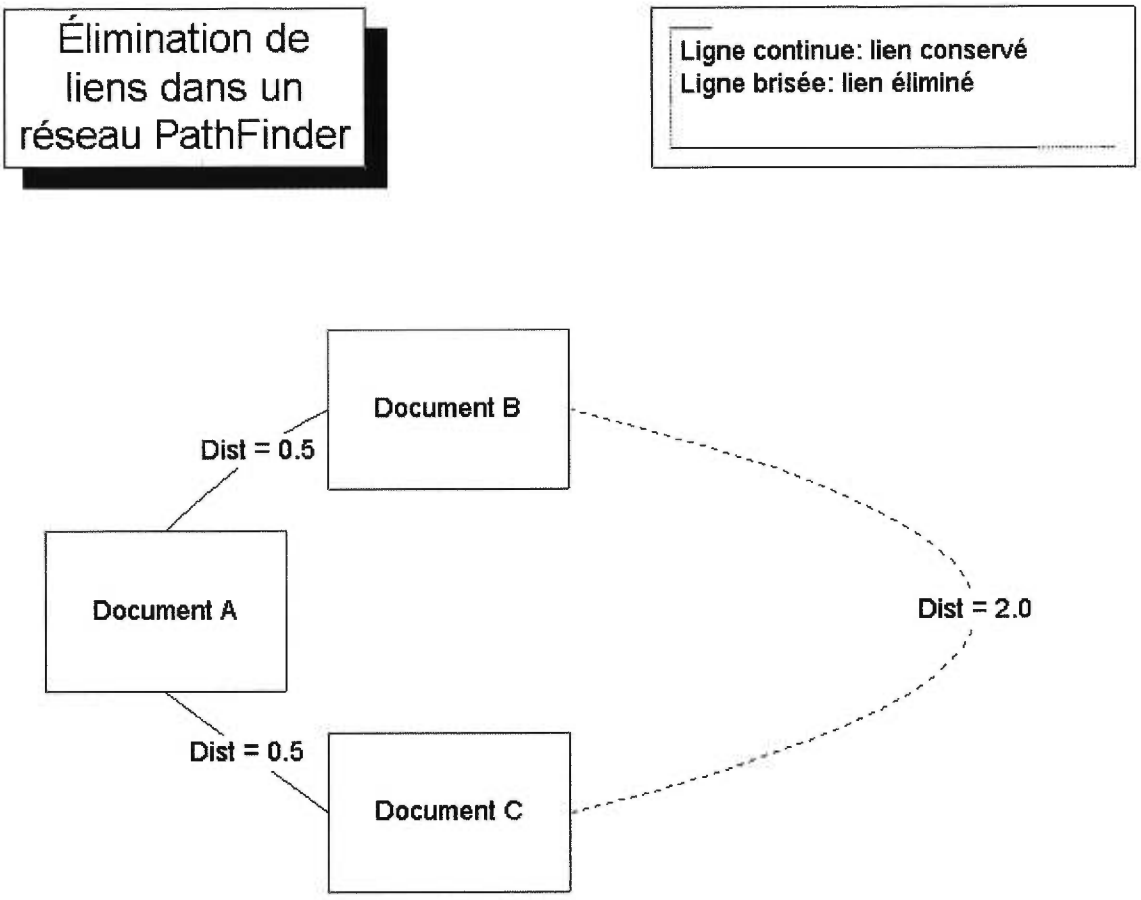

Figure 2-4: Élimination de liens dans un réseau PathFinder

L'approche est décrite dans un article où l'on raconte une expérience de construction d'un monde virtuel représentant les actes de trois conférences ACM CHI, regroupant 169 articles au total. Chaque article contient une liste de mots-clefs et ces mots-clefs sont utilisés pour calculer la ressemblance entre les documents. On obtient ainsi une matrice 169 x 169. En soumettant cette matrice à GSA, en passant aussi des paramètres qui informent l'algorithme, on obtient la base d'un monde virtuel. Ce monde sera ensuite représenté, en deux ou trois dimensions, en utilisant des méthodes d'affichage de graphes, discutées à la section suivante.

L'approche GSA et les autres approches d'auto-organisation nous apparaissent prometteuses car elles permettent de traiter de très grandes quantités d'information. Il serait fastidieux, voire impossible, de demander à un usager de spécifier manuellement une matrice 169 x 169. De plus, dans le cas de populations dynamiques de documents, l'usager ne pourrait avoir accès à de nouvelles sources d'information sans s'être d'abord plié à ce classement manuel.

2.3.2 Visualisation - graphes statiques et graphes dynamiques

Puisque la perception des relations entre les éléments d'un système est un outil de travail si important, la mise en page de modèles de telles relations a donné lieu à beaucoup de recherches. La plupart des systèmes existants utilisent des graphes pour représenter les structures relationnelles. Le problème se pose alors ainsi: il faut un algorithme qui assigne une position à chaque sommet et une route à chaque arête. Les techniques classiques procèdent en deux étapes: calculer une grande visualisation statique du graphe; puis permettre à l'usager de naviguer dans cette visualisation en tâchant de contrôler la complexité. Ces techniques ayant leurs limites, nous parlerons ensuite des approches dynamiques.

Modèles à base de ressorts

Les algorithmes de mise en page statique les plus utilisés à l'heure actuelle sont des modèles à base de ressorts. L'intention est de placer les noeuds de façon à minimiser une fonction d'énergie du système. Les noeuds sont représentés par des anneaux d'acier, les arêtes par des ressorts d'acier entre les anneaux. Le modèle original est basé sur les travaux de Eades puis de Kamada et Kawai [Huan98], [Fric99]. Sur chaque noeud s'applique - via les ressorts - la force d'attraction de tous les noeuds auxquels il est relié. Cette force suit la loi de Hooke: elle est proportionnelle à la différence entre la distance entre deux noeuds et la longueur du ressort au repos. Au temps t0, chaque noeud se voit assigner une position au hasard sur la carte. Puis, à chaque itération, les noeuds liés se déplacent progressivement l'un vers l'autre.

Le modèle est simple et fournit une mise en page raisonnable en termes de rapprochement des noeuds liés. Ce modèle est employé par le Classeur actif: on trouve à la figure 2-5 un exemple de saisie d'écran illustrant les résultats. Les noeuds reliés à un même parent vont tendre à s'étaler uniformément autour de ce parent. Toutefois, deux noeuds non-liés s'ignorent mutuellement; si le graphe est assez grand et en l'absence de contrainte supplémentaire, ces noeuds vont souvent occuper le même espace ce qui rend l'affichage confus. Une façon raisonnable de régler ce problème, en restant dans le domaine pseudo-physique, est l'addition de forces de répulsion entre les noeuds. C'est l'approche utilisée dans le système Narcissus, par exemple [Hend95].

L'approche dynamique à la présentation des graphes

À l'usage, on peut objecter deux choses à ce type d'algorithme: d'une part, le calcul et la navigation de graphes statiques atteint ses limites entre 100 et 1000 noeuds; d'autre part, elle est limitée à la présentation d'univers qui sont connus au démarrage. Elle est donc impossible à utiliser pour des univers mouvants et dynamiques, comme le Web par exemple [Huan98]. Les auteurs du système OFDAV - pour OnLine, Force-Directed, Automatic Visualisation - préconisent donc une approche dynamique à la présentation des graphes. Au lieu de montrer à chaque instant une carte de tout l'univers, ils en présentent une carte locale détaillée qui change avec les déplacements de l'usager.

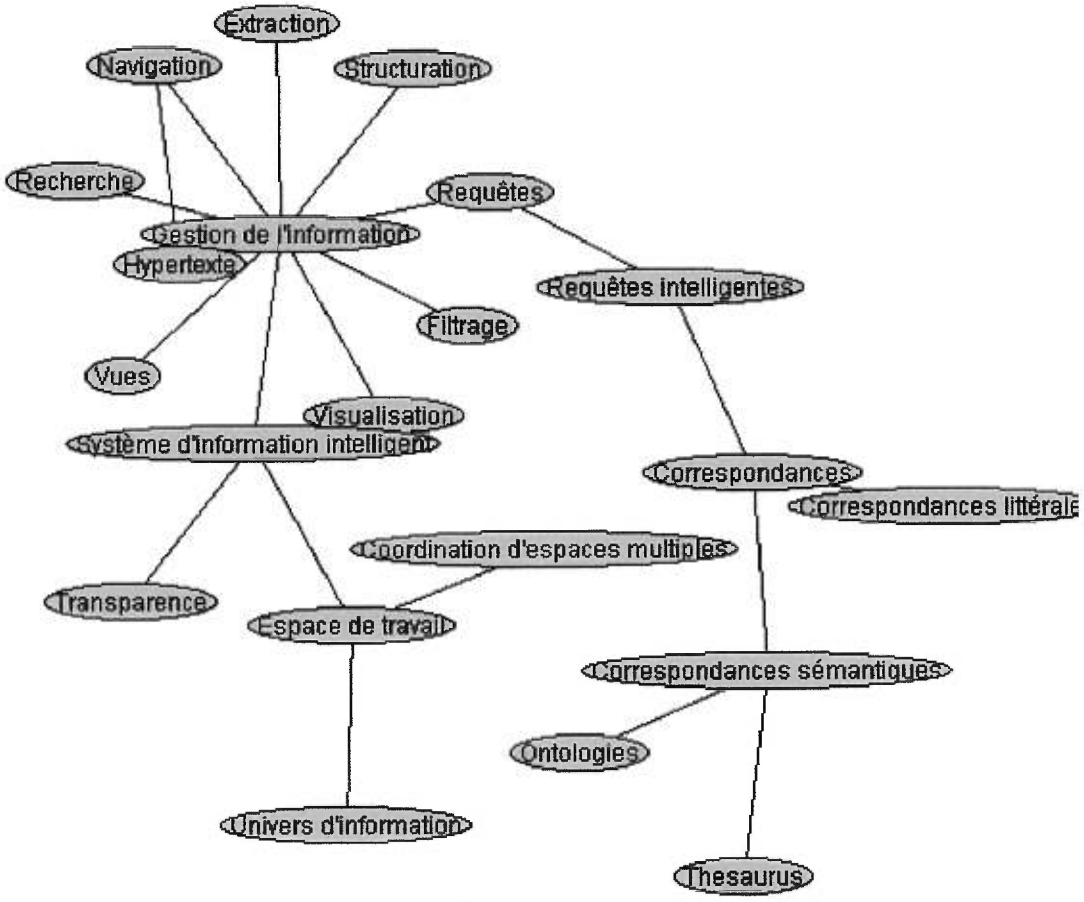

Figure 2-5: Saisie d'écran du Classeur illustrant le modèle à base de ressorts

Le concept de base dans cette approche est le cadre logique; l'exploration d'un grand graphe se fait en changeant de cadre logique. Chaque cadre comprend un sous-graphe connexe et une queue de noeuds-cibles. On fixe deux paramètres: la taille de la queue et la taille d'un voisinage. Dans OFDAV, avec une queue de longueur sept et un voisinage de taille un, on voit, pour un graphe dynamique du Web, entre 20 et 100 noeuds à l'écran à chaque instant. Le successeur d'un cadre est obtenu en choisissant un nouveau noeud-cible, en ajoutant son voisinage, et en choisissant au plus un noeud-cible pour le détruire, lui et son voisinage.

Tous les systèmes présentés jusqu'ici possèdent un certain degré d'auto-organisation ou d'apprentissage. Dans MIDS, on assiste à une structuration par grappes pour offrir une granularité plus fine; dans Syskill & Webert, on apprend un profil de l'usager; dans GSA, l'élément essentiel est l'auto-structuration de l'espace en réseau PathFinder. Nous parlerons, avec le dernier exemple, de systèmes dont le principal objectif est l'apprentissage: il s'agit avec NewsWeeder d'un apprentissage des préférences de l'usager dans le contexte des NewsGroups.

2.4 Apprentissage dans les systèmes de gestion d'information

Quiconque est abonné à un NewsGroup un tant soit peu actif reconnaîtra la difficulté d'y trouver l'information pertinente. Un outil appelé NewsWeeder [Mitc97] s'est attaqué à ce problème: il se concentre sur la modélisation de l'utilisateur pour fournir une interface plus conviviale en portant un jugement sur les documents susceptibles d'intéresser l'usager. Il utilise un réseau probabiliste, c'est-à-dire une structure capturant des relations de dépendance sous forme de probabilités. Le classificateur bayésien naïf correspond à une forme particulièrement simple de topologie de réseau probabiliste.

Le classificateur bayésien de NewsWeeder

Le créateur de NewsWeeder, Lang, utilise un apprentissage de type classificateur bayésien naïf [Mitc97]. Les algorithmes apprennent en traitant une collection de documents: ils créent un vecteur contenant environ 50.000 entrées, une pour chacun des 50.000 mots de l'anglais standard. *Il s'agira, dans un premier temps, d'entraîner le système en lui présentant des articles déjà classés. L'usager opère ce classement en fournissant un jugement allant de un (très intéressant) à cinq. Puis, on fait calculer au système plusieurs probabilités, qui seront ensuite utilisées pour classer de nouveaux articles.* Il faut

calculer, pour chaque classement C_i, la probabilité a priori de ce classement ainsi que la probabilité conditionnelle de rencontrer le mot w_j étant donné que le classement est C_i. Un nouveau document se verra assigner le classement C_i qui maximise la quantité suivante:

$P(C_i) * \text{Produit } P(w_j \mid C_i)$, où le produit est pris sur l'ensemble des mots w_j trouvés dans le nouveau document.

Les classificateurs bayésiens sont très rapides dans leurs prédictions [Pazz96]. Leur temps d'apprentissage est linéaire relativement au nombre d'exemples fournis, leur temps de prédiction est indépendant du nombre d'exemples et ils fournissent une granularité assez fine dans leurs calculs de probabilité. Une fois acquis le modèle des préférences de l'usager, NewsWeeder peut aussi attirer son attention sur des articles intéressants dans d'autres groupes. *Notons que pour l'expérience rapportée dans [Mitc97], l'usager a dû lire aux alentours de 2.000 articles pour entraîner le système.* NewsWeeder a eu un successeur plus évolué appelé WiseWire, disponible lui aussi sur le Web. Malheureusement, depuis son acquisition par Lycos, ce système est inaccessible.

2.5 Discussion

Ceci conclut notre tour d'horizon quant aux réalisations en gestion d'information sur l'Internet. Tous ces systèmes sont construits, ou bien sur de solides architectures (Harvest), ou bien avec un bagage théorique et pratique acquis par des groupes de chercheurs au terme de plusieurs années de travail. *Aucun de ces systèmes ne répond à tous nos besoins*: MIDS est colossal et privé; Syskill & Webert ne produit que des suggestions de navigation; NewsWeeder ne classifie que comme peu ou pas intéressant alors que nous désirons classer des documents comme pertinents pour tel ou tel concept. Quant à l'auto-organisation des documents, telle que présentée dans MIDS ou GSA, elle est certes intéressante en l'absence d'une structure de concepts explicite, mais nous disposerons justement de cette structure.

De tous ces exemples, les approches d'organisation de l'affichage sont à retenir, ainsi que la possibilité - démontrée par Syskill & Webert - d'obtenir des résultats raisonnables en formulant des requêtes intéressantes. Peut-on vraiment espérer construire un système d'enrichissement automatique d'un réseau sémantique, sans autre outil que les classes de la distribution Java et des techniques de RI? Il faudrait d'abord revoir et mieux formuler notre problème.

Chapitre III: Problème de l'enrichissement d'un curriculum

Dans le vaste contexte de la gestion d'information sur Internet, nous nous sommes posé le problème de *l'enrichissement automatique d'un réseau de concepts*. Nous avons examiné au chapitre II quelques systèmes qui nous paraissent représenter l'état de l'art pour la gestion d'information, examinant tour à tour les facettes de la cueillette, de la sélection et de la présentation de l'information. Le chapitre II nous a aussi permis de comprendre l'un des traitements sous-jacents commun à la plupart de ces systèmes, notamment celui de l'indexation d'une collection de documents et du calcul de la ressemblance entre une requête et un document.

Revenons maintenant au problème mentionné à la section 1.1. Nous rêvions d'un système qui puisse enrichir et modifier un réseau de concepts de façon dynamique, au moyen de recherche d'information sur Internet et d'assimilation de l'information trouvée au sein du réseau. Le problème n'est pas encore bien posé: s'agit-il d'un système qui "comprend" l'information recueillie? Si la réponse est oui, on parle d'un système analogue à Cyc dans son pouvoir d'assimilation; or Cyc est en cours de construction depuis des années déjà et a été estimé à une personne-siècle de développement [Guha90]. Sinon, *un système peut-il faciliter substantiellement la tâche de structuration des connaissances sans comprendre l'information?*

Il faut donc passer à un problème plus concret et mieux défini; c'est ce que nous ferons à la section un. À la section deux nous nous demanderons quelles sont les données exploitables pour résoudre ce problème. La section trois décrira notre plan d'attaque. Enfin la section quatre précisera le dernier maillon dans la chaîne de traitement du prototype.

3.1 Formulation et exemple

Supposons que l'on se contente d'une indexation thématique de nouveaux documents plutôt que d'exiger une indexation rhématique: le prototype devra découvrir de quoi traite un document mais non pas ce que le document dit à propos de son thème. Par exemple, il pourrait s'agir d'assister un travailleur intellectuel - professeur, étudiant, chercheur - en quête de ressources pour alimenter sa réflexion, d'une part, et cherchant aussi une façon souple et personnalisée de retrouver les documents qui l'intéressent. Pour fixer les idées encore davantage sans perdre trop de généralité, nous avons situé notre projet dans un contexte d'environnement logiciel de création et de gestion de systèmes tutoriels. Dans un tel environnement, on

rencontre un modèle plus ou moins explicite des connaissances dans un domaine, ainsi qu'une structure de ressources. D'une structure de concepts abstraite, nous passons à une structure de concepts telle qu'incarnée dans un squelette de cours. *Il s'agit de voir si, à partir d'un squelette de cours initial comportant un syllabus et quelques ressources textuelles déjà attachées à des éléments de ce syllabus, on peut enrichir la collection au moyen de recherches autonomes sur l'Internet.*

Supposons par exemple qu'un professeur prépare un cours sur la sémiotique et qu'il veuille étoffer chacun des chapitres de son cours au moyen de références accessibles électroniquement, dans le but d'aider ses étudiants à approfondir la matière. Il pourrait avoir un plan de cours comme suit (une représentation graphique et plus détaillée se trouve à la figure 3-1, page 31).

Cours XYZ: Introduction à la sémiotique

...

- Les signes

  Les signes selon Peirce (représentamen, interprétant, objet)

  cwis.kub.nl/~fdl/general/ people/baes

  Les signes selon Saussure: signifié, signifiant

...

- La modalité

...

- Les paradigmes et les syntagmes

  Analyse paradigmatique

  Analyse syntagmatique

...

Pouvons-nous envisager un système qui retrouve, sur l'Internet, divers documents pertinents et les rattache au bon endroit dans l'arbre des sujets esquissé plus haut? Nous avons essayé de faire ce travail manuellement en émettant une requête pour chacun des noeuds de cet arbre et en examinant les résultats: nous avons obtenu, avec Profusion, 60 réponses en moyenne pour chaque requête et nous avons péniblement examiné les sommaires, récupéré les documents qui paraissaient prometteurs et classé les documents dans l'arbre. Le travail est long et ennuyeux. Les documents se répètent. Certaines réponses sont farfelues. Nous reproduisons ci-dessous un tableau contenant dix des ressources examinées, avec des champs calculés manuellement pour chaque ressource.

Id	URL	Caractérisation	Traitement	Noeud
D1	www.semiotics.com/	Publicité	REJETER	
D2	www.qds.com/people/apope/ap_semiotic	Liens	Routine	Sémiotique
D3	www.aber.ac.uk/~dgc/ textan07.html	Liens	Routine	Sémiotique
D4	www.aber.ac.uk/~dgc/ semiotic.html	Tutoriel	Auto-référence	
D5	cwis.kub.nl/~fdl/general/ people/baes	Liens	Routine	Peirce
D6	www.epas.utoronto.ca:8080/french/as-	Journal	Créer un noeud	Sémiotique appliquée
D7	www.uwasa.fi/comm/termino/related/se	Journal	Routine	Sémiotique appliquée
D8	www.ahandyguide.com/cat1/l/l319.htm	Liens?		
D9	www.linguistlist.org/~ask-ling/msg01	Discussion	Routine	Eco
D10	oak.cats.ohiou.edu/~cs237495/semiotic	Liens		

Tableau I: Résultats partiels de l'expérimentation manuelle avec un cours sur la sémiotique

Décrivons brièvement les champs du tableau ci-dessus.

- Id: Un identificateur pour le document. Cela pourrait être le titre de la page sauvegardée localement.
- URL: Un Uniform Resource Locator pour le document. Il faut noter qu'il y a beaucoup de sites-miroirs.
- Caractérisation: Un effort de classement dans une taxonomie lâche de types de ressources.
- Traitement: Ce qu'il conviendrait de faire avec l'URL. *Routine* signifie simplement l'attacher à un noeud existant. *Auto-référence* signifie que l'URL en question était en fait le point de départ de notre squelette de cours. *Rejeter* est l'action effectuée quand les concepts représentés par le document ne trouvent pas leur place dans la hiérarchie, même à la racine; parfois, le rejet est une affaire de caractérisation.
- Noeud: Le noeud cible pour l'URL. Parfois il faudra le créer. Dans le cas de la sémiotique appliquée, on crée un noeud car on trouve plusieurs documents dont le titre contient cette expression, et on ne trouve pas un point d'attache bien clair dans la hiérarchie existante.

L'expérience nous a convaincu que l'automatisation de cette procédure fournirait certes un outil intéressant: il y avait assez de ressources pertinentes malgré tout pour que l'on veuille confier le repérage et le tri préliminaire à un logiciel. La figure 3-1 décrit sommairement le résultat escompté.

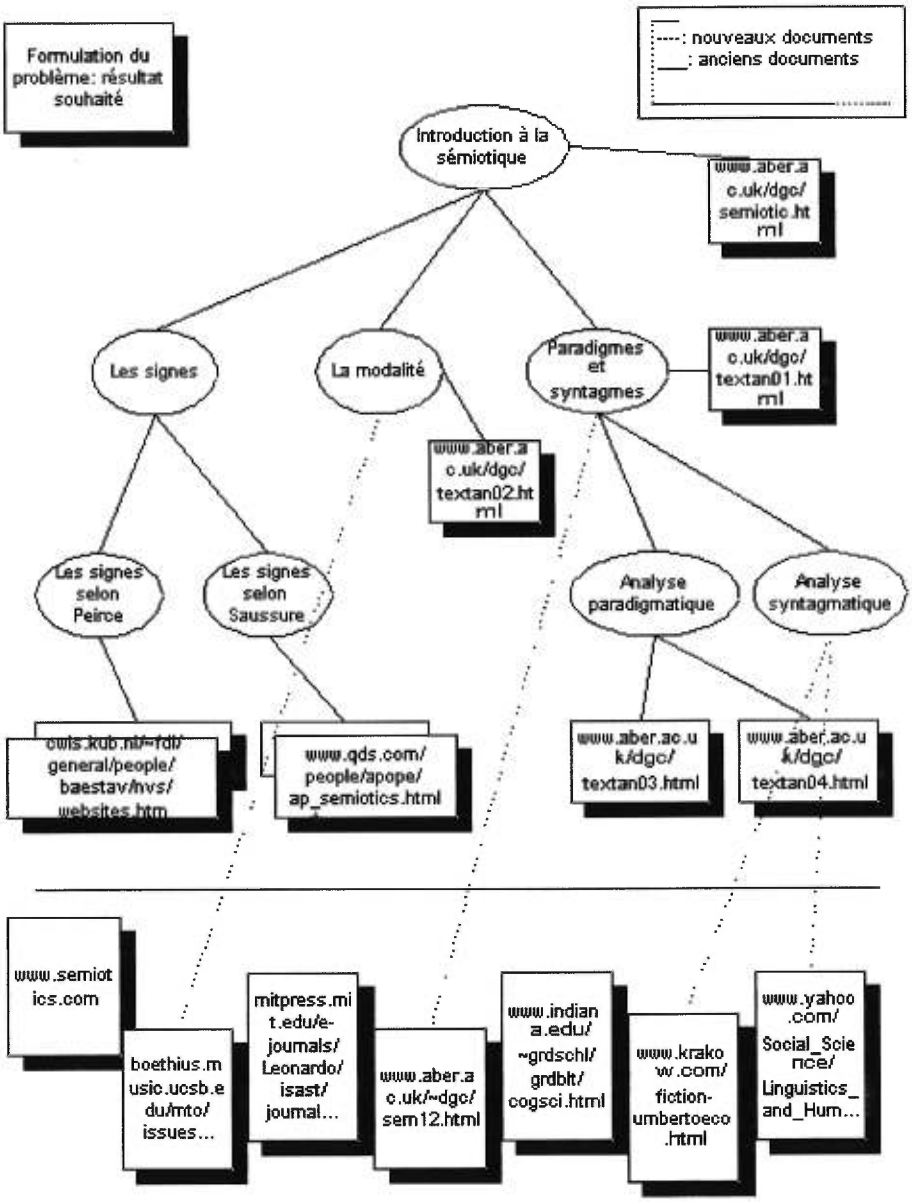

Figure 3-1: Formulation du problème - résultat obtenu manuellement

3.2 Les données

La seule donnée initiale de ce problème est un arbre de sujets, sections ou chapitres pour le cours ainsi que des références, des documents épinglés à certains des sujets. Cet objet ne possède a priori aucune

structure spéciale sauf qu'on peut supposer que c'est un fichier HTML, puisqu'il s'agit d'utiliser le Web comme infrastructure. On parle de squelette de cours parce que le document ne contient pas le cours lui-même mais seulement un genre de linéament, d'ébauche. Parmi les documents attachés, certains sont disponibles seulement sur un Intranet, les autres sont des documents publics accessibles sur l'Internet. Tous ces documents sont des fichiers HTML. Certains documents sont en français, les autres en anglais.

3.3 Ébauche de solution

Si l'on ne dispose, d'une part, que d'une structure de concepts et d'autre part que d'un univers hétéroclite de documents, le problème est trop difficile. La découverte de documents correspondant à ces concepts relève d'un modèle de connaissances du domaine (à tout le moins une ontologie), d'une part, et d'un modèle de traitement des documents faisant intervenir des connaissances profondes en langue naturelle, d'autre part. Cela nous amène dans le domaine de l'Intelligence Artificielle (IA). Par contre, dès que l'on fait l'hypothèse que certains documents sont déjà attachés à certains concepts, le problème change de caractère: au lieu d'une approche à base de connaissances, on peut envisager une approche statistique.

Cette hypothèse de documents attachés est raisonnable dans le contexte de notre problème d'enrichissement de corpus: en effet, ce corpus sera justement constitué d'un ensemble de ressources, dont plusieurs textuelles, qui sont en quelque sorte pré-indexées par leur association à l'un des noeuds du réseau. Insistons ici sur le fait que nous voulons construire un assistant intelligent, qui suggère des enrichissements mais ne travaille pas tout à fait automatiquement: cet assistant reçoit une partie de ses instructions implicitement, à travers la structure de noeuds et de liens créée par l'usager. Cet assistant ne peut fonctionner que parce qu'il dispose déjà de quelques données au départ.

3.3.1 Plan d'attaque

Les documents déjà attachés par l'usager peuvent nous fournir une passerelle pour circuler entre le monde des concepts et le monde des documents. En effet, on peut imaginer ce scénario:

- Pour chaque concept du réseau, on construit un modèle dans l'univers des documents; on obtient ainsi un "dictionnaire" global de termes représentatifs du réseau ainsi qu'une description de chaque concept à l'aide de ces termes représentatifs.
- Un agent émet une requête pour un concept en utilisant les termes représentatifs du concept.

- Supposons que le SRI invoqué ait renvoyé une liste d'URL. L'agent les examine un à un et produit un modèle pour chacun des documents repérés; ce modèle pourrait être simplement un vecteur des termes d'indexation avec leur poids. Ce pourrait être aussi un modèle plus riche, contenant des méta-données sur le document.
- Il faut maintenant associer le document à l'un des concepts. On doit évaluer la correspondance entre le modèle du document et les modèles de chacun des concepts. Ceci ressemble à une évaluation de pertinence.

Reprenons ici notre exemple de cours sur la sémiotique. Il s'agirait de remplacer les étapes faites manuellement dans notre première expérience par les traitements suivants:

1) Chaque chapitre ou sous-chapitre devient un noeud-concept dans notre modèle.
2) À partir des documents déjà attachés à des chapitres du cours, on va construire un index automatique: on aura ainsi une liste de termes avec leur poids pour chacun des chapitres et sous-chapitres. Chaque chapitre ou noeud se voit représenté par un vecteur de base.
3) Pour obtenir plus de documents sur un concept, on construit une requête, par exemple "Peirce signe" et on soumet cette requête à un SRI sur Internet.
4) Le SRI retourne une liste d'URL. Le système examine un à un ces documents. Il les indexe en utilisant les termes d'indexation retenus en 2). Il fait un calcul de ressemblance entre un nouveau document et chacun des noeuds de la structure existante.
5) Si la ressemblance maximale obtenue est sous un certain seuil, le document est rejeté comme étant du bruit. Sinon, le document est attaché au noeud qui semble le plus "proche".
6) *Parmi les documents retenus, certains seront ainsi rattachés à "Les signes selon Peirce" mais il est vraisemblable qu'on en trouvera plusieurs rattachés à "Les signes selon Saussure", ou encore "Les signes" ou même ailleurs. En général, le noeud source et le noeud cible pourront être différents.*
7) *Ces résultats seront considérés temporaires: ils seront présentés à l'usager comme tels et c'est à ce dernier que reviendra la décision. Mais au lieu d'une jungle de documents, on lui présente des documents déjà organisés selon son univers conceptuel du moment.*
8) Si l'usager accepte de classer le document X sous le noeud Y, on va reconstruire l'index, enrichissant ainsi la représentation du noeud Y pour la prochaine fois; on fera de même s'il choisit de modifier le classement, plaçant X avec Z.

3.3.2 Discussion

Il y a donc deux "processeurs" de RI dans ce modèle. Le processeur externe, n'importe lequel d'une liste de SRI supportés par notre système, reçoit une requête formulée automatiquement et tente de la satisfaire en fouillant l'univers ouvert des pages HTML sur le Web. Le processeur interne reçoit un document repéré par le processeur externe, le considère à son tour comme une requête et tente de satisfaire cette requête en fouillant son univers interne des noeuds de la structure de concepts avec leurs documents déjà attachés.

Le flot de données dans la figure 3-2 va d'abord de la gauche vers la droite: c'est toute la partie de la formulation et de l'émission de la requête. Puis on revient de la droite vers la gauche et c'est l'algorithme de classement des nouveaux documents. L'algorithme de classement peut donc être formulé ainsi: on choisit les termes significatifs des documents, ce qui nous fournit notre langage d'indexation, les vecteurs formant la base de notre espace; on indexe la collection, ce qui produit un ensemble de vecteurs dans cet espace, un pour chaque concept; on indexe le nouveau document en utilisant le langage d'indexation obtenu; on fait le calcul de ressemblance entre ce nouveau vecteur et tous les anciens; le vecteur le plus ressemblant fournit le classement cherché.

Le problème d'enrichissement automatique donne donc prise au traitement: c'est essentiellement un problème de RI classique et c'est bien ainsi que nous entendons l'attaquer. Pourtant, dans notre dernier diagramme, il y a une zone encore floue à l'interface entre les deux moteurs. Ce point reste incertain: nous avons parlé d'émettre une requête à un SRI et d'examiner les résultats. Il nous faut maintenant analyser cette étape du traitement.

3.4 Découverte de documents sur Internet

Nous voulons construire un outil qui puisse enrichir de façon semi-automatique une collection existante de documents. L'une des facettes importantes de ce travail consiste en la cueillette, sur l'Internet, de documents qui risquent d'être pertinents. Comment trouver ces documents? Les systèmes de repérage d'information déployés sur Internet existent déjà, ils sont légion. Il n'est donc pas nécessaire d'en construire un: nous construirons simplement une requête pour un SRI existant. Il faudra analyser les

entrées et les sorties des requêtes à des SRI existants. Il faudra choisir un moteur de RI. Il faudra communiquer avec ce moteur de RI.

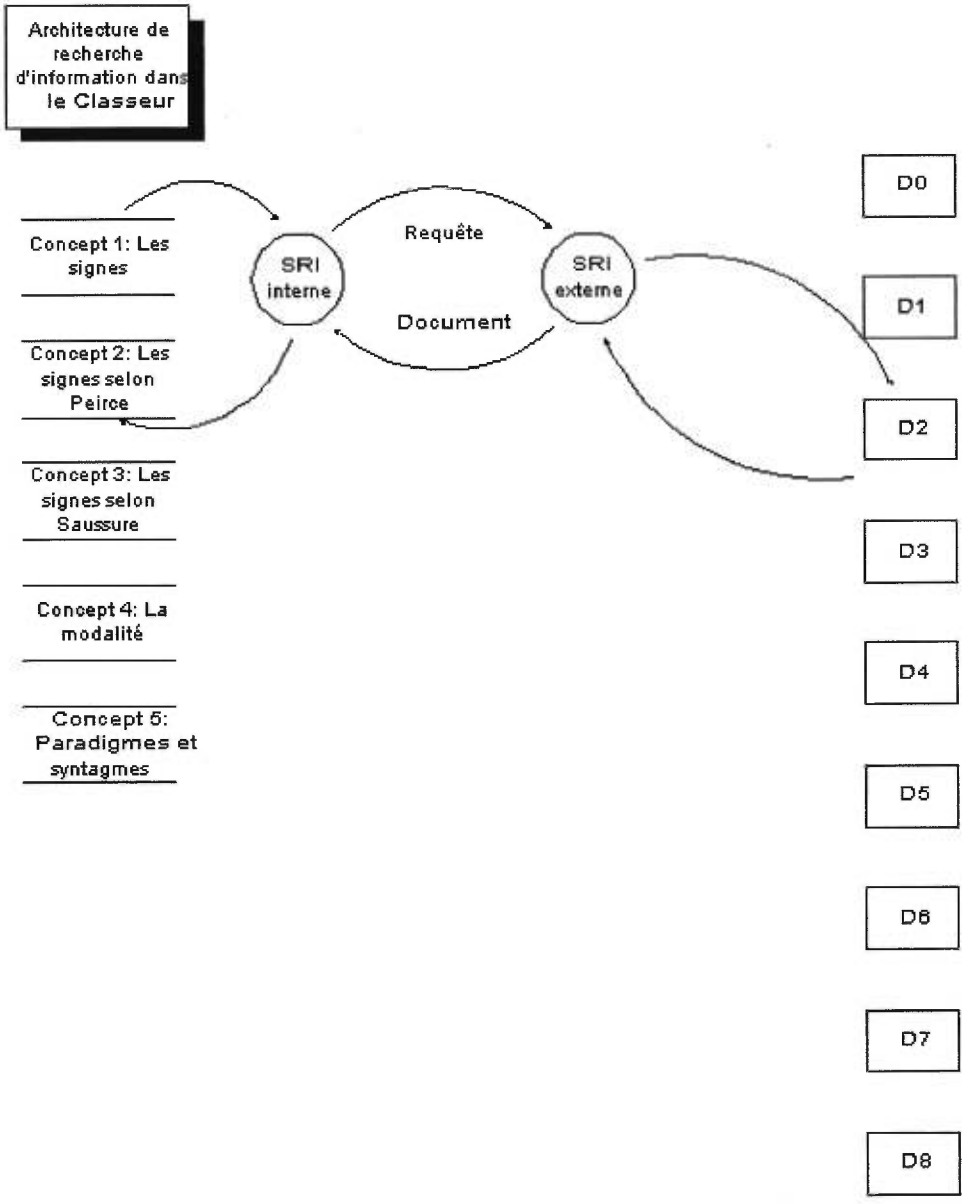

Figure 3-2: Les deux moteurs de RI dans l'architecture du Classeur actif

3.4.1 Interaction avec un service de recherche d'information sur Internet

Le standard pour les interactions entre un client et un serveur sur le Web est *CGI*, le Common Gateway Interface. Un script CGI est un programme qui produit dynamiquement une page HTML. L'information

envoyée par l'usager est fournie au script et les sorties du script sont renvoyées à l'usager. La syntaxe des URL permet de lancer un script CGI tout aussi bien que d'ouvrir une page HTML simple. Il suffit donc, pour interagir avec un serveur sur le Web, d'utiliser un langage qui supporte les URL et d'apprendre quelle est la syntaxe des questions et des réponses pour le serveur en question. Concentrons-nous sur l'analyse des questions et des réponses à un moteur ou un service de RI sur Internet.

Il suffit de construire un URL qui pose la question et d'ouvrir cet URL: ceci correspond à poser la question et recevoir la réponse. C'est le mécanisme utilisé par les fureteurs quand on fait la recherche en mode interactif, donc en épiant les interactions au niveau CGI on peut comprendre comment interroger tel ou tel engin de RI. Malgré de vastes différences, il y a un dénominateur commun qui permet à une interrogation "simple" de procéder toujours de la même façon: <adresse> <préfixe> <terme1> <séparateur> ... <termeN> <options> <suffixe>. Quant à la page retournée par l'engin, elle contient des URL pointant vers des documents jugés pertinents, mais elle contient aussi de la publicité et divers liens locaux. Il faut analyser cette page pour en extraire une liste de documents potentiellement pertinents.

Nous nous sommes d'abord concentrés dans ce projet sur *Lycos*, *AltaVista* et *Profusion*. Par contre, comme nous l'expliquerons au moment de la discussion de nos classes Java, l'implantation permet d'interroger d'autres systèmes à volonté si l'on connaît les formats d'entrée/sortie. Vu les déménagements fréquents, cette souplesse semblait importante. Nous en avons fait l'essai récemment en intégrant l'accès au serveur *Inference* sans avoir à modifier le prototype autrement que par l'ajout d'un fichier descriptif.

3.5 Conclusion

Nous avons esquissé notre plan d'attaque en reformulant chaque étape comme une opération de RI. Le survol de la RI fait au chapitre II nous a muni des éléments nécessaires pour l'indexation et nous venons de régler le dernier détail préliminaire en examinant comment interroger un SRI sur Internet; ceci conclut notre analyse. Il s'agit maintenant de voir comment nous assemblerons ces éléments. *Nous appellerons notre système un Classeur actif ou tout simplement, un Classeur; c'est un classeur parce qu'il permet de ranger les documents et de les organiser; il est actif parce qu'il va à la recherche de documents sur Internet.* Il est temps de procéder à une modélisation conceptuelle complète du Classeur. Une fois construit ce modèle, nous passerons à la description du prototype et des résultats obtenus.

Chapitre IV: Modélisation conceptuelle du Classeur

Nous avons esquissé un système appelé Classeur actif, qui permettrait d'enrichir une structure de concepts de manière semi-autonome. Nous avons toutefois formulé cette description au niveau des résultats escomptés ainsi que d'un plan d'attaque encore peu détaillé. Il serait bon de construire un modèle conceptuel. *Ce modèle nous servirait entre autres à faire l'inventaire des difficultés et à nous assurer que nous disposons des données et connaissances requises pour chaque traitement.* Diverses approches méthodologiques s'offrent à un concepteur quand il s'agit d'analyser un système: les plus populaires à l'heure actuelle sont les méthodologies orientées-objet, en particulier UML (Unified Modelling Language) [Mull97].

Pourtant, j'ai choisi de procéder à la modélisation avec KADS (Knowledge Acquisition and Design System). Ce choix peut surprendre: il m'a été dicté par une expérience professionnelle en cognitique qui m'a permis de manipuler très facilement ces modèles, de comprendre qu'ils répondent parfaitement aux besoins d'un concepteur et finalement d'apprécier leur facilité de lecture pour un non-initié. Ce chapitre commence par une courte introduction à KADS. Le reste du chapitre présentera le modèle le plus utile pour notre inventaire, i.e., le modèle d'expertise et en particulier le modèle d'inférences: nous passerons donc à une description en termes de trois modèles d'inférence KADS [Wiel91].

4.1 La méthodologie KADS

La méthodologie KADS de construction de systèmes à base de connaissances (SBC) représente une approche fructueuse à la modélisation conceptuelle, qui va bien au-delà des SBC. Développée dans le cadre du projet ESPRIT en Europe, c'est une façon de travailler qui prend en compte les multiples facteurs risquant d'avoir un impact sur l'analyse, la conception, le développement et la mise en service d'un système d'information. KADS est une approche d'ingénierie: elle décrit un SBC non pas comme un contenant que l'on remplit en extrayant les connaissances d'un expert, mais comme un modèle opérationnel qui doit présenter certains comportements [Wiel91].

Lorsque l'on utilise la méthodologie KADS pour construire un système intelligent, on produit plusieurs modèles; ils sont énumérés dans le tableau II. Ces modèles, du modèle organisationnel au modèle de design, sont reliés par des transformations. Le modèle d'expertise, qui représente l'apport méthodologique

le plus important de KADS, est à son tour composé de quatre modèles - connaissances, inférences, tâches et stratégies.

Modèle	**Objectif**
Organisationnel, application, tâche	Problème, organisation, attentes, contexte, contraintes.
Expertise - Connaissances	Objets, structures, propriétés, valeurs, relations
Expertise - Inférences	Opérations cruciales, connaissances en entrée et sortie
Expertise - Tâches	Inférences atomiques intégrées avec contrôle
Expertise - Stratégies	Niveau de méta-planification
Coopération	Transfert d'information entre système et usagers
Conceptuel	Intègre coopération et expertise
Design	Le modèle le plus physique

Tableau II: Les modèles dans la méthodologie KADS

L'une des prémisses de cette décomposition est qu'il est utile de faire la distinction entre différents types de connaissances selon les rôles que ces connaissances sont appelées à jouer. L'autre prémisse est que l'on peut organiser ces types en couches aux interactions limitées de manière à faciliter la réutilisation de certains sous-modèles: par exemple, si l'on arrive à spécifier les inférences en termes de rôle des entrées-sorties plutôt que de leur nature, on pourra reprendre un modèle d'inférence développé pour diagnostiquer des problèmes logiciels et l'appliquer dans un domaine totalement différent. Parmi ces quatre sous-modèles, nous nous concentrerons sur les modèles d'inférence: c'est là que réside la plus grande complexité du problème qui nous occupe.

4.2 Modèles d'inférence

Le modèle d'inférence dresse la liste de toutes les opérations "intelligentes" d'un système, avec les connaissances requises en entrée et les connaissances produites en sortie. Un tel modèle possède une dimension graphique importante, qui est un graphe bipartite. Dans ce graphe, les ellipses contiennent des verbes et les rectangles contiennent des noms. Chaque verbe représente une opération qui manipule un ou plusieurs objets de connaissances pour en produire un seul en sortie; un exemple serait Segmenter, dans la figure 4-1. Chaque nom représenterait normalement un rôle joué par des objets de connaissances,

plutôt que de nommer des objets du domaine: j'ai jugé ce degré d'abstraction inutile dans le contexte, puisque nous pouvons ignorer ici les questions de réutilisation.

Nous présentons un modèle d'inférence pour les opérations suivantes:

- l'analyse des données et la construction des structures internes;
- la construction, l'émission et le traitement de requêtes;
- le classement d'un document dans la structure de concepts.

Pour chaque modèle d'inférence, nous incluons une représentation graphique et nous poursuivons avec une discussion sommaire de des opérations. Notons que les modèles d'inférence se situent à un niveau strictement analytique: aucun effort n'est fait pour spécifier l'ordre d'exécution des inférences. Cette couche de contrôle, dans l'approche KADS, est plutôt dévolue au modèle de tâches.

4.3 Construction des structures internes à partir du squelette de cours

Le point de départ des opérations du prototype est un squelette de cours contenant les titres des chapitres et des sous-chapitres ainsi que des URL attachés à certaines de ces rubriques. Tel quel, ce squelette est impropre à supporter l'enrichissement car il ne démontre aucune structure directement utilisable. Nous allons donc le lire et l'interpréter selon une syntaxe et une sémantique fort rudimentaires qui suffiront pourtant à construire les structures de données en mémoire vive jugées nécessaires pour la formulation des requêtes et surtout pour le classement des nouveaux documents. Notre approche sera de construire, à partir du fichier initial, des fichiers auxiliaires qui représentent l'interprétation du système.

La figure 4-1 révèle quatre inférences, c'est-à-dire quatre opérations: segmenter, indexer, analyser1 (produire les structures hiérarchiques) et analyser2 (produire les structures relationnelles). Chacune de ces opérations est décrite brièvement dans ce qui suit.

Segmenter un curriculum

- **Entrées:** Un fichier contenant un curriculum; des connaissances sur la syntaxe HTML;
- **Sortie:** Une liste d'objets possédant une certaine cohérence, couvrant une ou plusieurs "lignes";

- **Traitement:** On supposera certaines conventions minimales, comme le fait que les concepts soient balisés par H1, H2 ou H3. Cette étape, fort simple, réduit considérablement la complexité des suivantes. Il s'agit essentiellement d'un regroupement préliminaire tenant compte des balises et sauts de lignes. On obtiendra ainsi, par exemple, une liste telle que: (<H1, "Internet - IFT 3220: Préliminaire">, <Neutre, "Jan Gecsei">, <H2, "A. Internet">, <H3, - "Historique de l'Internet, croissance, statistiques, futur: I-2">, <URL, "http:// wwwbacc. ift.ulaval.ca...>, <H3, ...>). Cette liste formera la matière première des traitements subséquents.

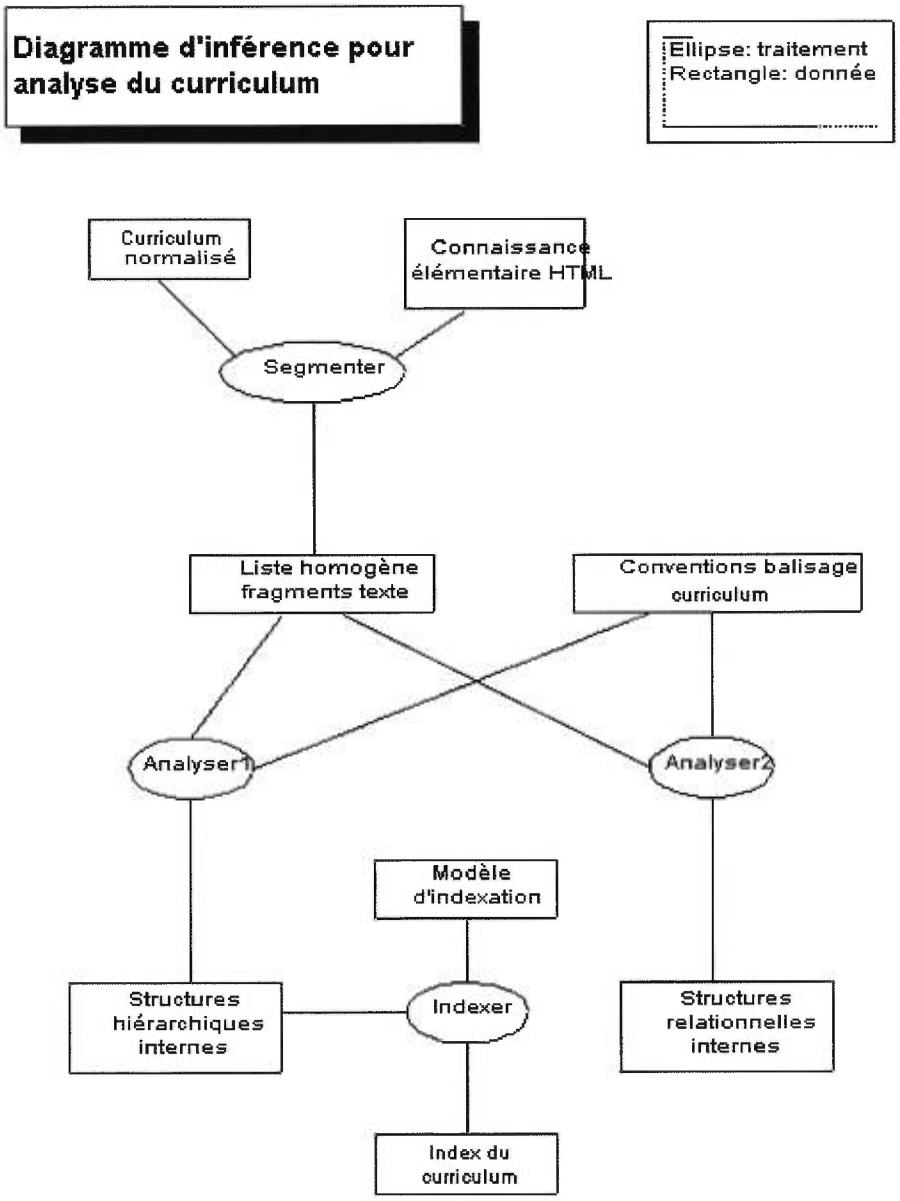

Figure 4-1: Construction des structures internes à partir du squelette de cours

Produire les structures hiérarchiques

- **Entrées:** Une liste de fragments textuels, des conventions syntaxiques;
- **Sortie:** Une structure hiérarchique de sujets, de fragments textuels neutres et de documents reliés;
- **Traitement:** Les fragments non-balisés sont attachés au plus bas niveau qui les précède. L'usage de H1, H2 et H3 est interprété comme une hiérarchie de sujets. On traverse la liste de fragments en utilisant une pile pour construire la hiérarchie. La figure 5-5 au chapitre V fournira plus de détails.

Produire les structures relationnelles

- **Entrées:** Une liste de fragments textuels, des conventions syntaxiques;
- **Sortie:** Une liste de liens entre concepts, et entre concepts et documents;
- **Traitement:** Ce traitement ressemble au précédent, sauf que les éléments hiérarchiques des rapports entre deux objets sont ignorés à ce stade. La structure est surtout utilisée pour faire du graphisme ou de la navigation.

Produire les structures d'indexation

- **Entrées**: Documents, points d'attache, modèle d'indexation;
- **Sortie**: Une structure décrivant la collection avec les termes et leur poids pour chaque concept;
- **Traitement**: Tel qu'expliqué au chapitre II, un analyseur lexical extrait une liste de termes candidats pour les documents de la collection, on produit et analyse des statistiques sur les termes, on sélectionne un ensemble de termes d'indexation et on assigne des poids à ces termes.

Après ces quatre opérations, les préparatifs sont terminés et le Classeur actif devrait pouvoir fonctionner. Passons donc aux requêtes.

4.4 Construction, émission et traitement de requêtes

Il s'agit maintenant de comprendre mieux comment va se produire cette chaîne de traitement qui prend pour point de départ un des concepts et qui se termine au moment où des documents ont été téléchargés pour être confiés au dernier modèle, celui du classement. *Nous décrivons les inférences suivantes, qui*

forment un genre de pipe-line dans la figure 4-2: formuler une requête logique, la transformer en requête physique pour un SRI donné, lancer la requête, recevoir la réponse sous la forme d'une liste d'URL et enfin trier la liste et récupérer les documents jugés intéressants. Notons que nous employons le qualificatif logique ici pour indiquer le langage de l'usager, et physique pour le langage du SRI.

Formuler une requête logique

- **Entrées**: Corpus, réseau sémantique, ressources linguistiques;
- **Sortie**: Une représentation logique de la requête, logique parce que plus près du langage de l'usager;
- **Traitement**: Il s'agit de traduire un besoin d'information, tel qu'exprimé par un noeud de la structure de concepts, en une liste de mots qui seront utilisés par une autre opération pour construire la requête physique. On pourra choisir des termes dans les noms ou des termes spécifiés par l'usager.

Formuler une requête physique pour un SRI

- **Entrées**: Requête logique, choix d'un SRI, données spécifiques pour chaque SRI;
- **Sortie**: Une représentation de la requête sous la forme utilisée par l'API (précisée au chapitre V);
- **Traitement**: La traduction de la requête logique est simple: il suffit de connaître la syntaxe utilisée par un service donné. Cependant, il faut tenir compte de l'évolution des SRI: dans les méta-connaissances, on envisage une mise à jour occasionnelle des caractéristiques syntactiques.

Lancer la requête

- **Entrées**: Requête physique;
- **Sortie**: La liste d'URL retournée par le SRI;
- **Traitement**: La requête est faite à l'intérieur de cette inférence; en sortie nous obtenons une liste d'URL. La requête physique contient protocole, hôte et demande de service: en ouvrant cet URL avec CGI, on émet la requête et reçoit les réponses. L'émission de la requête est donc relativement facile.

Trier la liste d'URL

- **Entrées**: La liste d'URL, leurs caractéristiques externes, certaines données administratives;
- **Sortie**: Une liste réduite d'URL;
- **Traitement**: Le SRI produit dynamiquement une page HTML qui contient, parmi la publicité et d'autres non-informations, une liste des URL jugés aptes à satisfaire le besoin d'information. On peut déjà tenter d'en évaluer l'utilité en examinant les dates, une liste négative ou une liste positive.

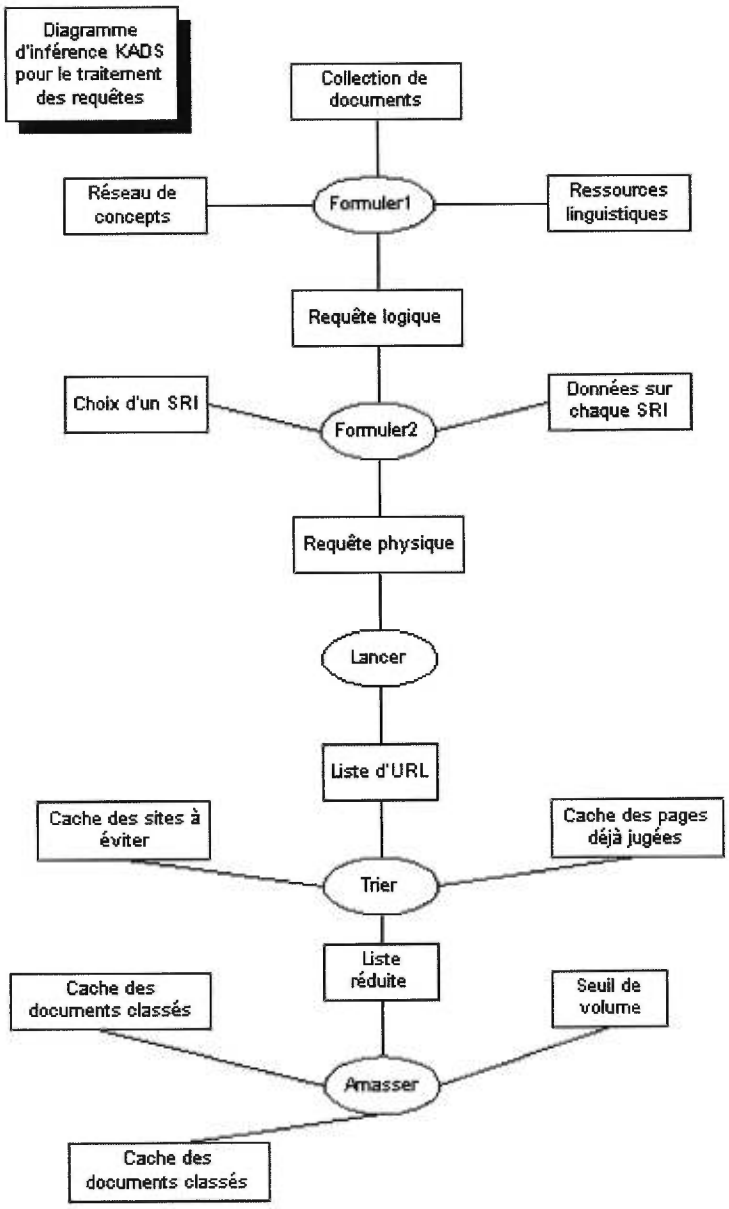

Figure 4-2: Diagramme d'inférence KADS pour la construction et le traitement des requêtes

Amasser les documents

- **Entrées**: Liste d'URL, méta-connaissances, seuil de volume;
- **Sortie**: Les documents plein-texte placés dans une cache;
- **Traitement**: la seule difficulté ici est de déterminer où et quand télécharger les documents, et de respecter les seuils spécifiés par l'usager. Il faudra aussi créer un nom local pour chaque document sauvegardé localement, ainsi que garder la trace de ces paires < nom local>< nom global >.

Les deux modèles d'inférence qui précèdent ont permis au Classeur actif de digérer le contenu d'un squelette de cours et d'aller sur l'Internet se procurer d'autres documents susceptibles de s'intégrer dans cette structure de concepts et de documents. Comment allons-nous procéder à cette intégration? Les techniques de RI (chapitre II) sont ici d'un grand secours.

4.5 Classement d'un document dans le réseau

Au terme du cheminement proposé par les deux modèles d'inférence qui précèdent, on dispose d'un ou de plusieurs documents à intégrer dans la structure de concepts. La requête avait été formulée dans le but de trouver des documents pour le noeud X, le SRI externe semble indiquer que le document est pertinent pour le noeud X; c'est maintenant le tour de notre SRI interne de déterminer où ce document va le mieux s'insérer dans la structure. Nous présentons les inférences suivantes: analyser chaque document repéré, suggérer une classification et intégrer le document compte tenu des décisions de l'usager.

Analyser chaque document repéré

- **Entrées**: Texte du document, stratégie d'indexation, ressources linguistiques;
- **Sortie**: Nouvelle représentation du document;
- **Traitement**: Notre modèle de document est basé principalement sur l'indexation. Les décisions sont prises conjointement avec les décisions sur la représentation du corpus; *la représentation choisie ici devra avoir deux propriétés importantes: supporter la visualisation et supporter la comparaison avec les concepts du curriculum.*

Suggérer une classification

- **Entrées**: Représentation du document, structure de concepts, méta-connaissances;
- **Sortie**: Recommandation pour la classification du nouveau document;
- **Traitement**: Il faudra comparer deux points dans un espace et faire intervenir une métrique de proximité. On se contentera de correspondances textuelles, en comparant des vecteurs produits automatiquement, dans un modèle, et des vecteurs de termes d'indexation manuelle dans l'autre.

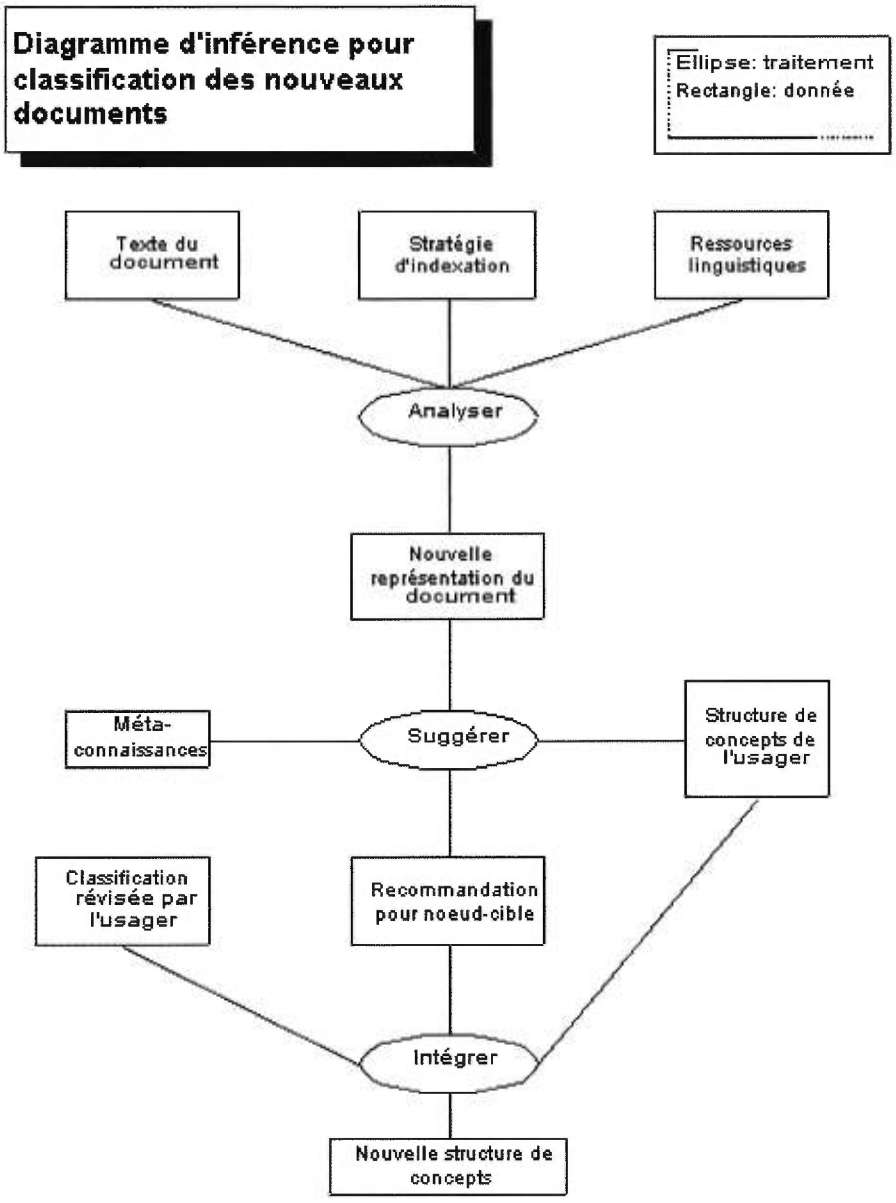

Figure 4-3: Diagramme d'inférence pour la classification des documents

Intégrer le document dans le corpus

- **Entrées:** Réseau de concepts originel, classification proposée, classification révisée;
- **Sortie:** Le réseau, mis à niveau en tenant compte du document;
- **Traitement:** En bout de ligne, il faut intégrer le nouveau document à la collection de l'usager et le rendre accessible à travers le noeud classificateur. Mais le système sera d'autant plus intéressant qu'il tentera de modifier ses critères de classification en tenant compte du jugement de l'usager.

4.6 Discussion sur les modèles d'inférence

Les modèles d'inférence décrits dans les trois dernières sections suivent le formalisme KADS à ceci près qu'au lieu de montrer, en entrée et en sortie, des méta-classes, c'est-à-dire des jetons représentant essentiellement des rôles de connaissances, nous avons incorporé directement les objets du domaine. Les méta-classes sont utiles dans KADS en tant que niveau d'indirection qui permet de réutiliser un modèle d'inférence dans un domaine différent. Dans notre contexte, par contre, comme nous n'avons pas présenté de modèle de connaissances statiques et comme ce prototype est un projet isolé, cette entorse à la méthodologie nous a paru souhaitable.

Le chapitre III avait tracé un plan d'attaque pour ce problème d'enrichissement d'un curriculum. Au chapitre IV, nous venons de procéder à un exposé plus détaillé des traitements à supporter dans le Classeur. *Nous avons choisi pour ce faire la méthodologie KADS, en particulier ses modèles d'inférence qui ont permis de cataloguer et de préciser les traitements confiés au Classeur actif.* Au terme de cette analyse, il semble que nous disposions d'assez d'éléments pour passer à la conception et l'implantation d'un prototype. Le prochain chapitre décrit notre implantation.

Chapitre V: Un prototype pour le Classeur actif

Nous venons de décrire les traitements du Classeur actif à un niveau orienté-connaissances, c'est-à-dire que nous avons procédé comme si nous devions écrire un SBC; nous voulions ainsi nous assurer que tous les éléments de solution étaient compris. Il s'agit maintenant d'implanter ces traitements avec un langage orienté-objet. Dans cette optique, nous nous tournons vers la modélisation physique d'un prototype capable d'enrichir un curriculum au moyen de recherche sur Internet. Nous parlerons architecture et mécanismes; nous présenterons les principales décisions d'implantation; nous décrirons quelque peu l'interface, puis les classes du problème, c'est-à-dire collection, concept et une requête.

5.1 Cas d'usage, objets et mécanismes

L'analyse conceptuelle du chapitre IV nous amène à distinguer trois mécanismes principaux:

1. la production des fichiers internes à partir du curriculum initial fourni par le concepteur
2. le repérage de nouveaux documents sur le Web, jugés pertinents au sujet du curriculum
3. le classement de ces nouveaux documents dans l'arbre des sujets du cours.

Ces mécanismes servent à implanter les cas d'usage du Classeur et ces cas d'usage, à leur tour, représentent un raffinement de la formulation du problème telle que présentée au chapitre III. Dans le paradigme orienté-objet, les mécanismes sont réalisés en créant des objets qui coopèrent pour atteindre des objectifs. Nous présenterons ici les principaux objets du Classeur actif et leurs interactions majeures pour illustrer comment les mécanismes du chapitre IV prendront forme. *La présente section constitue un dernier tour d'horizon du modèle logique; les artefacts qu'elle décrit sont le résultat d'une reformulation du problème en termes d'objets. À la section suivante, on se tourne vers les décisions d'implantation.*

Énumérons d'abord les opérations que l'usager peut faire avec ce Classeur.

5.1.1 Les cas d'usage du Classeur actif

Les cas d'usage ont été introduits dans la méthodologie orientée-objet par Ivar Jacobson. Ils décrivent le comportement d'un système du point de vue de l'usager, en termes d'actions et de réactions. Ils permettent de définir les frontières d'un système et les échanges entre ce système et son environnement. Un cas d'usage représente un type d'utilisation du système; il décrit la fonctionnalité activée en réponse à une stimulation exercée par un acteur externe au système [Mull97]. Énumérons les cas d'usage du Classeur, représentés à la figure 5-1.

- Cas d'usage CU1 - Enrichir concept: l'usager choisit un concept et demande au système de repérer des documents pertinents pour ce concept.
- Cas d'usage CU2 - Classer document: l'usager choisit un document et demande au système de déterminer le meilleur concept pour y attacher ce document.
- Cas d'usage CU3 - Ajuster classification: l'usager demander au système de rattacher le document X au concept Y; les structures internes seront affectées.
- Cas d'usage CU4 - Construire index: l'usager demande au Classeur de procéder à l'indexation automatique de la collection existante. Si le fichier d'index existe, il est simplement chargé en mémoire.
- Cas d'usage CU5 - Examiner document: l'usager choisit un document et demande au Classeur de le lui laisser explorer.
- Cas d'usage CU6 - Éditer concept: l'usager désire modifier les données relatives à un concept, en particulier les termes d'indexation manuelle.
- Cas d'usage CU7 - Éditer réseau: l'usager désire modifier la structure de son curriculum en ajoutant des concepts ou des liens, ou en en retranchant. Ce cas n'est pas supporté à l'heure actuelle.

La liste de ces cas d'usage découle assez naturellement de la manière dont le problème a été défini au chapitre III. Nous préciserons quelque peu chaque cas lors de la description des menus et fonctions, à la section 5.3. Les cas d'usage illustrés dans ce diagramme suggèrent déjà certaines classes: concept, document, index, par exemple. Nous passons à la présentation des objets retenus et des relations qu'ils entretiennent.

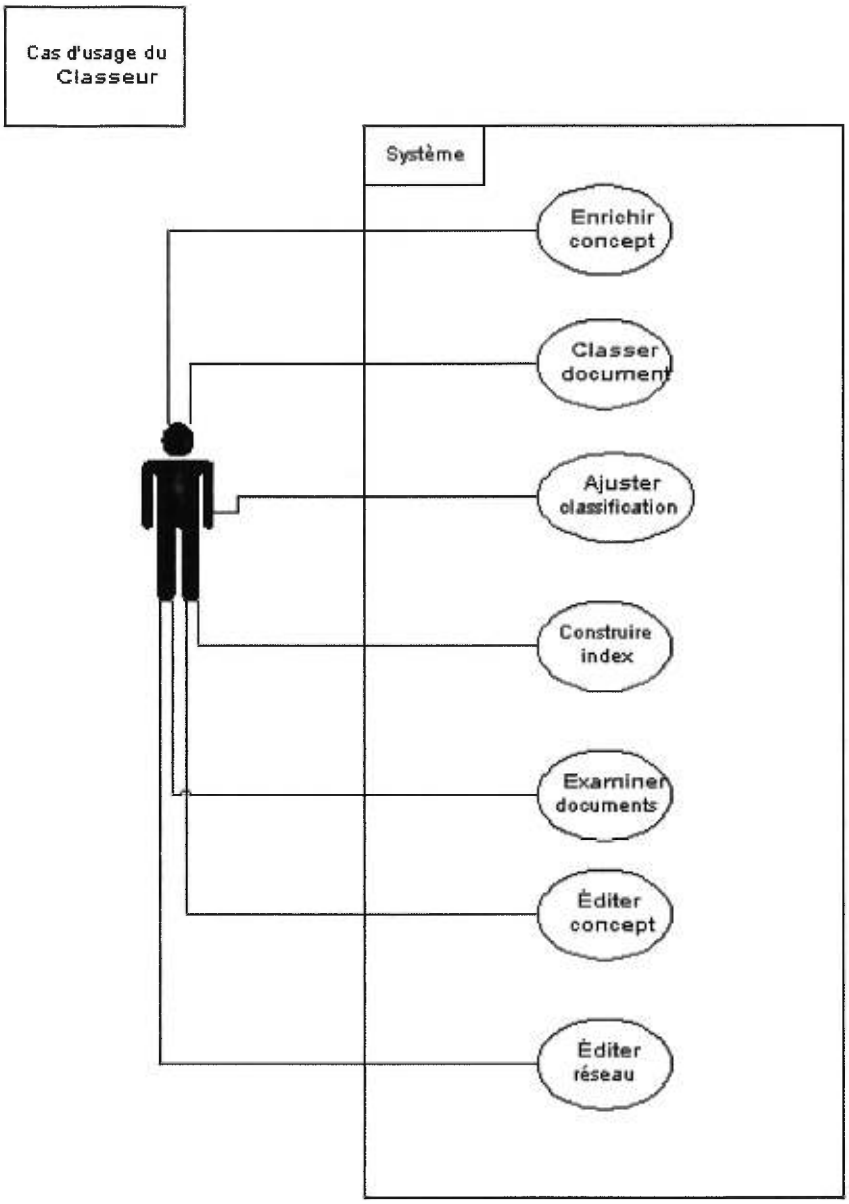

Figure 5-1: Les cas d'usage du Classeur actif

5.1.2 Les principaux objets du prototype

Les principaux objets de notre implantation - nous les désignerons du nom de leur classe - sont la *Collection* et les *Requêtes*. (Nous utilisons les italiques pour dénoter les objets et les classes). Dans ce qui suit, le terme Classeur acquiert une nouvelle signification: c'est un objet qui permet l'accès graphique à la fonctionnalité d'enrichissement et de classement.

La figure 5-2 présente les classes les plus importantes dans leurs rapports avec l'usager et le monde extérieur. La *Collection* regroupera et intégrera des objets tels que concept, document et index alors que les *Requêtes* permettront de réaliser le cas d'usage CU1, Enrichir Concept. Le curriculum rédigé par l'usager sera transformé par la Collection, ou l'un des ses auxiliaires, en des structures plus faciles d'utilisation. Puis, l'usager pourra émettre des requêtes en les adressant à la classe RequêteClasseur. Enfin, c'est encore la Collection qui veillera au classement des documents recueillis.

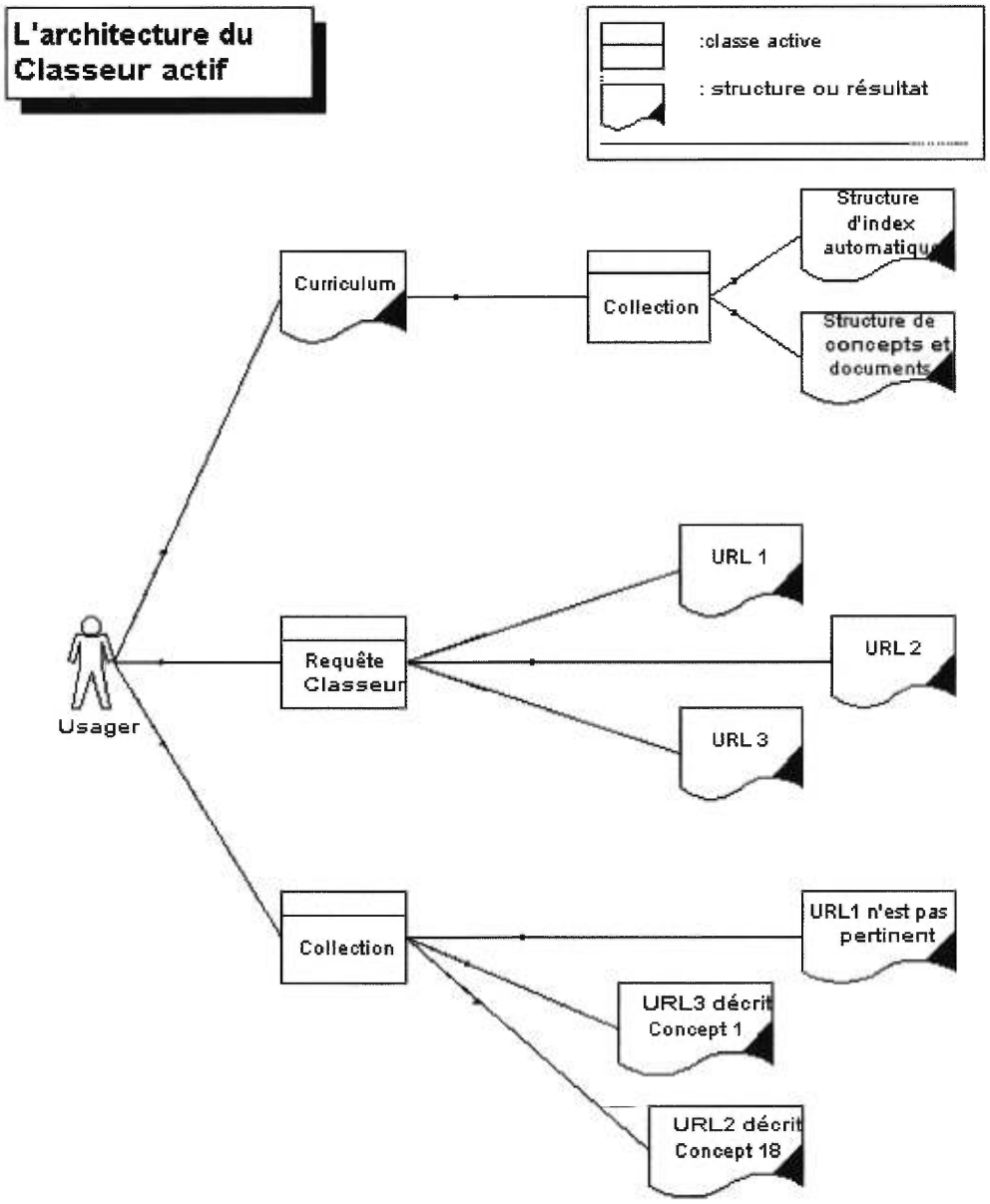

Figure 5-2: Les classes du Classeur dans leur contexte d'utilisation

Vu de l'extérieur, les principales classes sont donc la *Collection* et la *RequêteClasseur*. Mais ces classes ne peuvent accomplir leur travail qu'au moyen d'une décomposition en objets et relations plus spécialisés. On trouvera dans la figure 5-3, ci-dessous, les principaux objets du prototype.

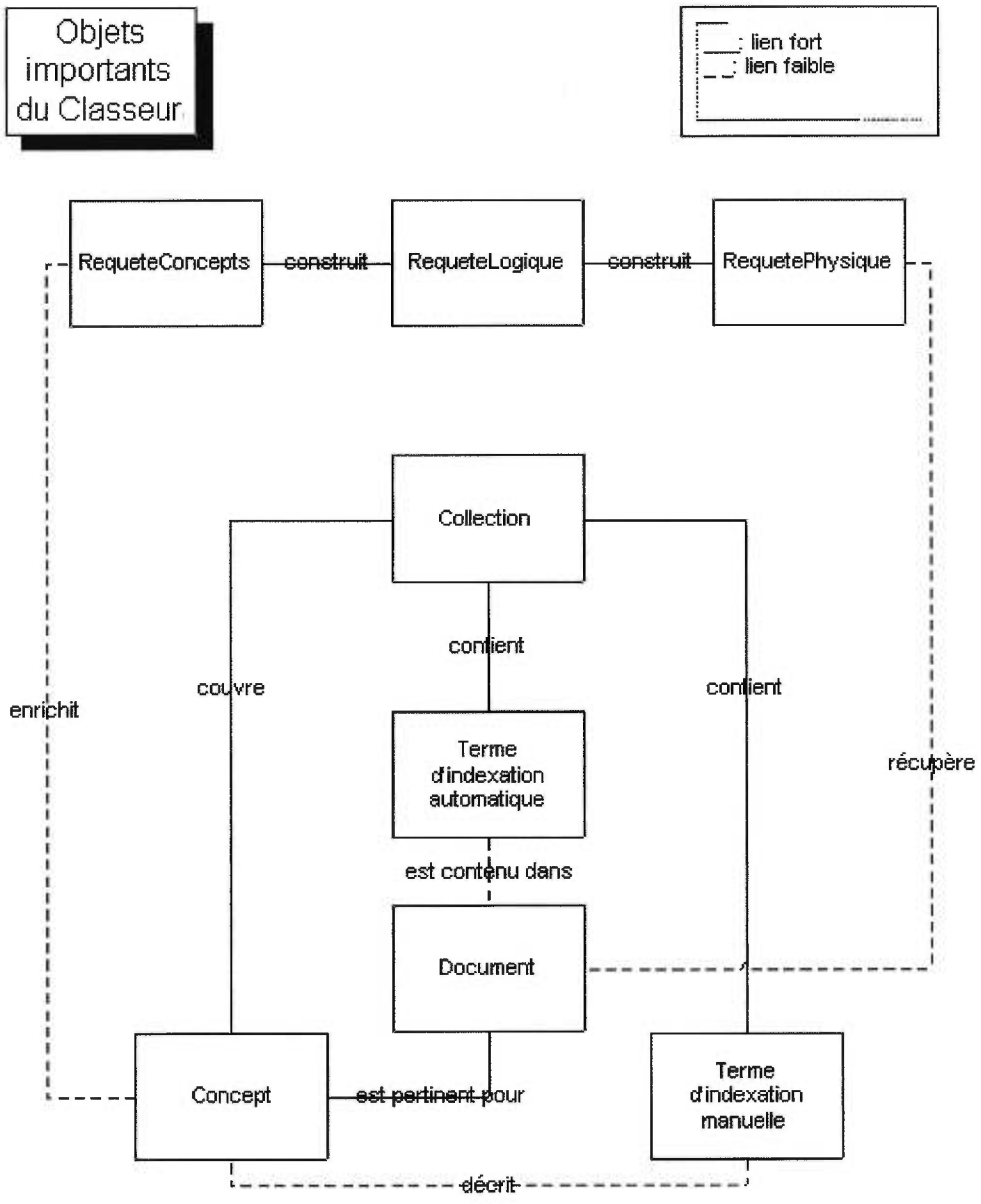

Figure 5-3: Les objets importants

Le Classeur permet l'accès à un objet *Collection*. La *Collection* contient un tableau de *Concepts* qui correspondent - dans le curriculum original - à des fragments textuels spécialement marqués; un tableau de *Concepts* représente un cours et ne devrait donc pas contenir plus de quelques dizaines de *Concepts*. La *Collection* contient aussi un tableau de *Documents*; ce sont les documents déjà jugés pertinents par l'usager et confirmés par lui comme appartenant à un *Concept* donné. À un niveau d'implantation, la *Collection* contient aussi un tableau de termes d'indexation automatique et un tableau de termes d'indexation manuelle, tous les deux du même type, *NoeudIndex*. Ces objets assurent toute la fonctionnalité d'indexation et de comparaison. (Dans notre terminologie d'implantation, influencée par l'affichage des objets et de leurs relations sous forme de graphe, un noeud désigne n'importe quel objet affichable, que ce soit un terme d'indexation, un concept ou un document).

Pour les requêtes, nous avons les *RequêteConcepts*, *RequêteClasseur* et *RequêtePhysique*, dont le mécanisme sera expliqué dans un instant. La *RequêteClasseur* correspond à ce que nous avons appelé plus haut le niveau logique, c'est-à-dire qu'elle s'intéresse à certains termes. Chaque *RequêteClasseur* construit et utilise un objet *RequêtePhysique*. La *RequêtePhysique* encapsule les connaissances qui permettent de communiquer avec un SRI particulier pour obtenir des documents concernant ces termes. Enfin la *RequêteConcepts* se situerait, dans une implantation plus évoluée, à un niveau abstrait, celui des concepts plutôt que des mots. Elle construit des *RequêtesClasseur* pour un concept donné.

Nous venons de présenter les principaux objets du Classeur de façon statique, en termes de leur contenu et de leurs liens. Nous pouvons maintenant décrire les mécanismes ébauchés plus haut, mais cette fois en termes plus physiques. Nous décrirons la façon dont les objets collaborent pour obtenir les résultats voulus.

5.1.3 Le mécanisme d'analyse du curriculum et de construction des structures internes

Nous avons décrit plus haut (section 4.3) la construction, à partir d'un curriculum produit par un usager, des structures internes du prototype. Mais cette présentation, faite sous la forme d'un modèle d'inférence KADS, était restée à un niveau plus conceptuel. Nous passons ici à un niveau de conception détaillée.

Qu'est-ce qu'un curriculum? Quelle est sa syntaxe? Quelle est sa sémantique? On doit pouvoir répondre à toutes ces questions pour passer d'un tel objet aux structures internes utilisées par le prototype dans son traitement. Nous ne croyons pas qu'il existe à l'heure actuelle un standard, informatique ou autre,

pour décrire un curriculum. Cette situation est liée au fait qu'il existe peu ou pas d'outils de support à la construction de cours. Le domaine de recherche le plus actif semble être plutôt le développement d'éditeurs multimedia, dont certains encapsulent des principes de conception pédagogique. On a déjà fait allusion à cette lacune au chapitre I.

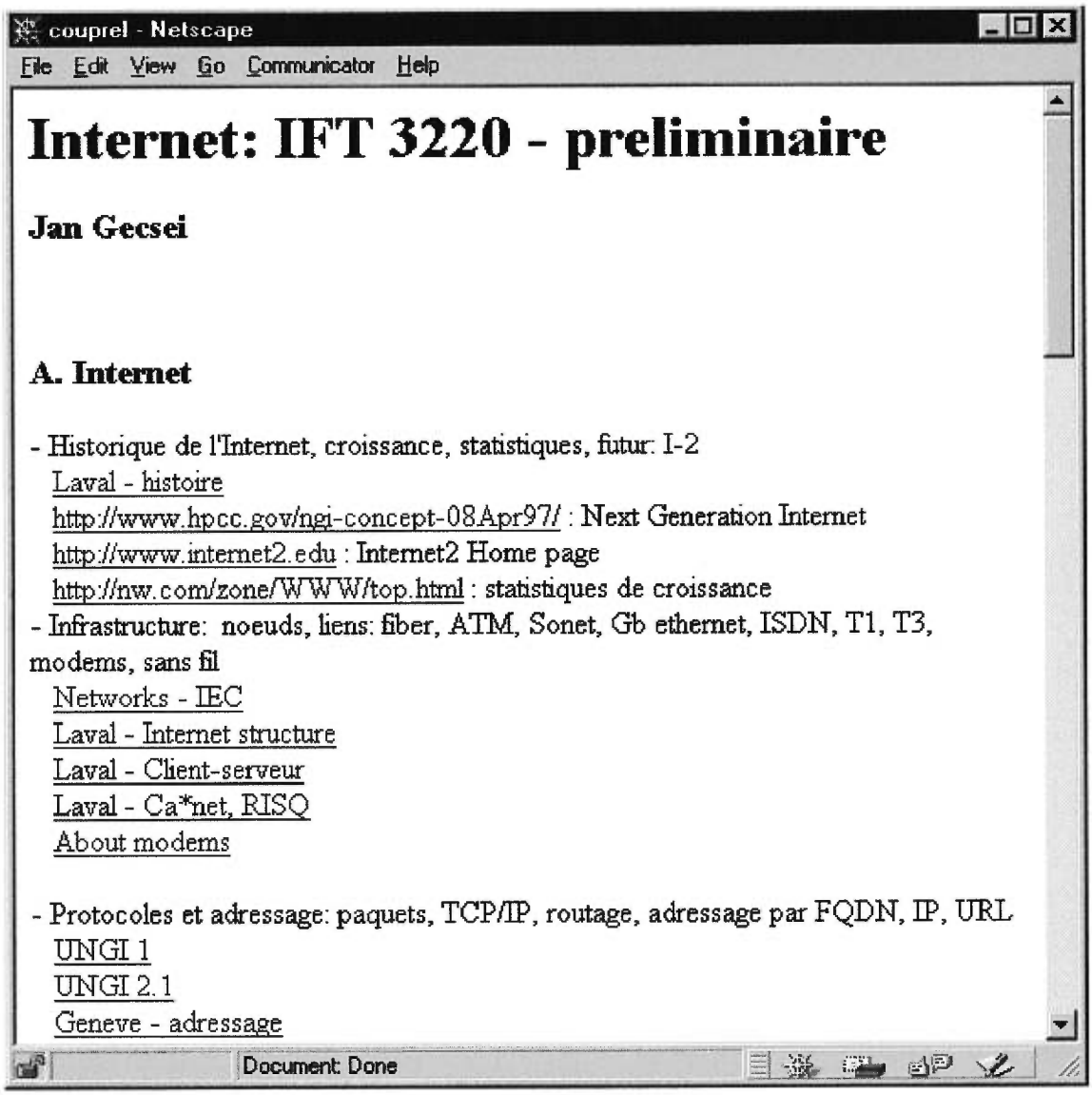

Figure 5-4: Début du curriculum original utilisé dans l'expérimentation

Nous avons mis l'accent sur le texte: le curriculum serait un texte avec une structure simple. Il serait toujours temps d'importer ou d'exporter un curriculum quand les formats externes, binaires ou textuels, deviendraient mieux définis. Pourtant il fallait imposer une structure, faute de quoi la construction des structures internes devenait un problème d'analyse de langue naturelle. Nous avons étudié le curriculum

exemple et avons tenté d'adapter et de généraliser. Le curriculum exemple était une ébauche produite pour un premier cours sur l'Internet à l'Université de Montréal, ébauche produite par le professeur Jan Gecsei.

Il aurait été difficile d'analyser ce curriculum dans son état initial; les seules balises utilisées étaient H1 pour <Internet: IFT 3220 - préliminaire> et H3 pour <Jan Gecsei> et pour le niveau un, c'est-à-dire, par exemple, <A. Internet>, <B. Web - Principes et utilisation>. Le seul autre moyen de structuration était un caractère, le tiret. Plutôt que d'écrire du code d'analyse pour transformer le tiret en indicateur hiérarchique, j'ai préféré spécifier des normes de balisage pour un curriculum et j'ai normalisé le curriculum original pour qu'il suive ces normes. *Un curriculum normalisé est donc un objet hiérarchique qui déclare sa hiérarchie au moyen de balises HTML de la façon la plus simple qui soit: le titre est balisé par Title, le premier niveau est balisé par H1, le second niveau par H2 et ainsi de suite.*

Rappelons qu'il s'agit d'induire un arbre de concepts à partir d'un curriculum écrit dans un style hiérarchique faible. Au terme d'une analyse syntaxique sommaire, le Classeur produira un arbre de concepts. *Il procède en deux étapes; au cours de la première étape, une étape de segmentation, il produit un tableau de fragments textuels*, de type *TexteAnnoté*. Chaque fragment a été produit de façon à tenir compte des sauts de lignes et des balises; il possède une certaine intégrité et sera considéré par la suite comme un objet atomique. *Une fois créé ce tableau, on le réexamine pour créer un arbre d'objets de type NoeudConcept. Cette étape consiste en une analyse syntaxique classique à base de pile.* La *Collection* essaie d'abord de lire une structure de concepts, c'est-à-dire une liste de concepts avec mots-clefs et documents attachés, telle que produite par l'interprétation du curriculum. Si elle ne la trouve pas, elle la produit (voir figure 5-5). Puis elle lit ou produit la structure de termes d'indexation automatique.

Fichiers produits en sortie - Sauvegarde locale des URL

Ce mécanisme produit une structure de concepts éditable, une structure de liens, éditable elle aussi, ainsi qu'un fichier inversé contenant les termes d'indexation automatique et leur poids pour chaque concept. La structure de liens consiste simplement en une liste de triplets, <Nom du premier objet, Nom du second objet, Nom du lien>. La structure de concepts contient un nom pour chaque concept, une liste d'URL ainsi qu'une liste de termes d'indexation manuelle, produite automatiquement au départ mais destinée à être pondérée et remaniée par l'usager. Cette liste sera le principal outil de rétroaction du prototype.

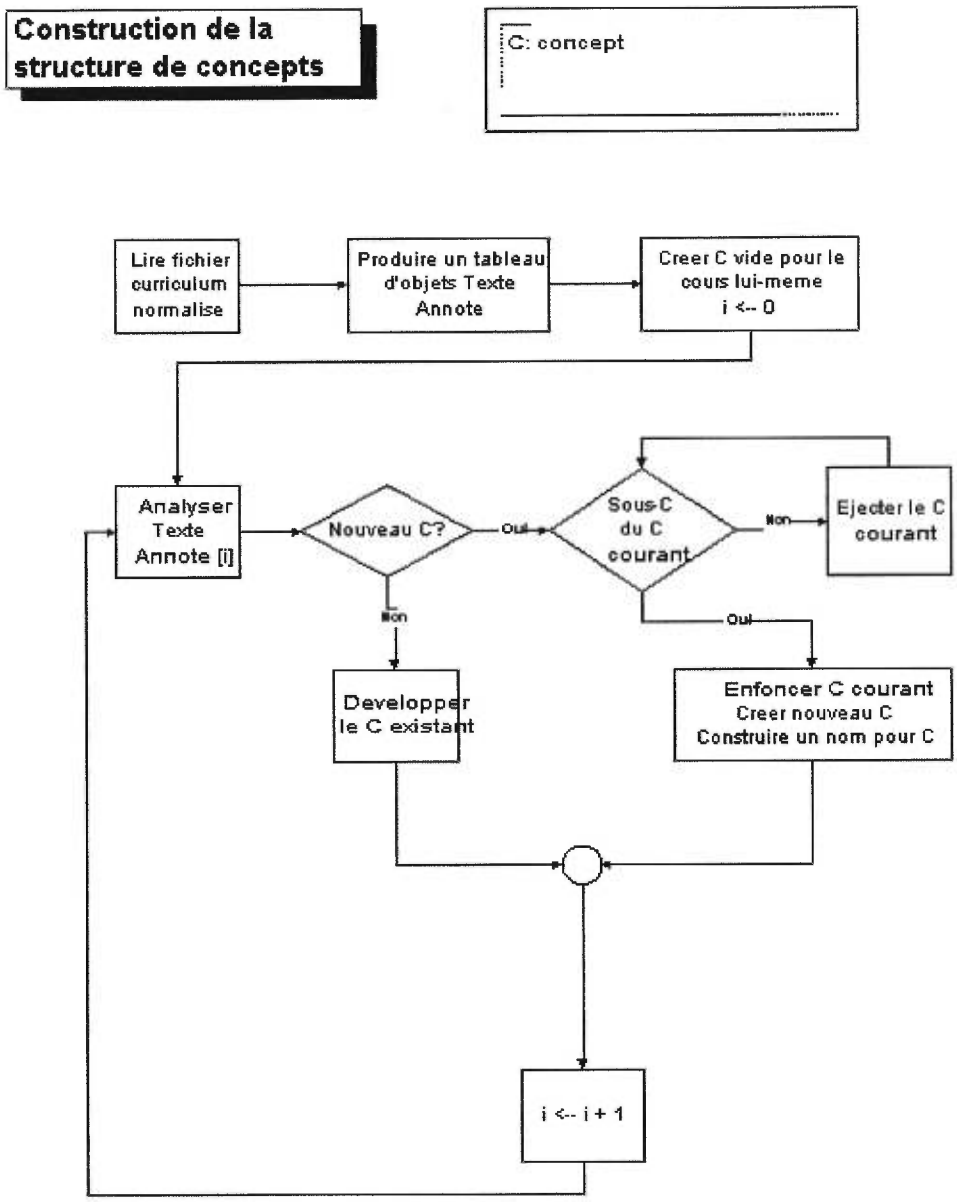

Figure 5-5: Mécanisme d'analyse du curriculum

Les documents attachés à la collection sont d'abord sauvegardés localement. Ils reçoivent alors un nom ad hoc, et la correspondance entre ce nom local et leur nom global, leur URL, est consignée dans un fichier décrivant la cache. Au moment de l'indexation de la *Collection*, ils sont supposés exister localement. Mais après cette indexation, l'usager peut les détruire. Il pourra quand même y accéder, au besoin, à travers la fenêtre de furetage qui, elle, demandera à la cache comment retrouver le document. Ces mesures permettent de maintenir l'utilisation de l'espace-disque au minimum.

Le deuxième mécanisme, celui du repérage des documents, n'est que faiblement lié aux structures internes que nous venons de construire; par contre au niveau du classement, décrit ensuite, cette intégration deviendra indispensable.

5.1.4 Le mécanisme de repérage

Le mécanisme de repérage fait intervenir trois classes principales: *RequeteConcepts*, *RequeteClasseur* et *RequetePhysique*. Bien qu'encore peu utilisée, parce que non-intégrée avec l'IU (Interface Usager), la classe *RequeteConcepts* offre le plus de fonctionnalité des trois. Elle a pour objectif de recueillir un certain nombre de documents pour un concept donné, et tente d'atteindre cet objectif en créant et en émettant des *RequetesClasseur* pour le concept en question, modifiant la formulation et le SRI externe jusqu'à ce que son seuil soit atteint. La *RequeteClasseur* est un objet intermédiaire, qui sait comment manipuler une *RequetePhysique* et qui porte sur certains termes en particulier. Les étapes sont les suivantes:

- *Un noeud est choisi pour l'enrichissement et un objet de la classe RequêteClasseur est alors créé. Cet objet compose sa requête au niveau conceptuel d'abord, en utilisant les mots contenus dans le nom du noeud. Un système de RI externe est choisi pour l'émission de la requête.*
- *Les données requises pour interroger ce SRI sont stockées dans un fichier ASCII externe, nommé <SRI>.Dat, par exemple AltaVista.Dat. Pour se lancer, la RequêteClasseur crée un objet de la classe RequêtePhysique. La RequêtePhysique construit un URL. L'acte d'ouvrir cet URL donne lieu à l'interaction cherchée.*
- Une fois l'URL de la requête ouvert et rempli par le SRI, l'objet *RequêtePhysique* l'examine pour en retirer les URL. Un certain filtrage doit se faire à ce moment, puisque plusieurs des URL constituent en fait de la publicité ou encore des liens à des pages locales. Ce filtrage est basé sur l'élimination des adresses locales ainsi que des adresses contenant certaines chaînes de caractères spécifiques à chaque service de RI.
- Si l'URL survit au filtrage initial, la *RequêtePhysique* en fait la demande en l'ouvrant, crée une copie locale du texte pour fins d'indexation et d'examen par l'usager, et procède à une mise à jour des données de la cache.

Il ne reste plus qu'à classer les documents ainsi obtenus.

5.1.5 Le mécanisme de classement

Le classement des documents est basé sur l'indexation, dont nous avons déjà parlé au chapitre II. Nous ne ferons ici que reformuler en employant cette fois les noms de nos classes.

Classement du document par le système

La classe Collection encapsule les connaissances nécessaires à la représentation du document, la représentation de la collection et le calcul de ressemblance. Cette ressemblance est basée sur un calcul de projection dans un espace vectoriel, tel qu'élaboré plus haut (section 2.1). Précisons que deux calculs de ressemblance sont effectués, l'un avec l'index produit automatiquement et l'autre avec l'index des termes fournis par l'usager pour chaque noeud. Chacun des ces index est constitué d'un tableau d'instances de la classe *NoeudIndex*.

Ajustement du classement par l'usager et rétroaction

Après avoir examiné un document à travers l'objet *Fureteur*, l'usager peut choisir de confirmer ou modifier le classement apporté par le système: ceci fait intervenir la classe *Collection*, qui utilisera pour ce faire des objets *Concept*, *Document* et *NoeudIndex*, ainsi que divers objets d'interface usager. La *Collection* procède à une rétroaction implicite en modifiant, lors de la prochaine indexation, les poids des termes dans les objets *NoeudIndex* qui constituent l'index automatique. Elle procède de plus à une rétroaction explicite en augmentant, dans les *NoeudIndex* de l'index manuel, les poids des termes partagés par le *Document* et son nouveau *Concept* d'attache.

5.1.6 Discussion

Nous avons introduit quelques grandes classes et mécanismes pour le Classeur. Mais tout cela reste un peu théorique. Comment allons-nous passer à l'implantation? Il y a plusieurs décisions à prendre - langage, plate-forme, format - nous allons les présenter une à une. Nous continuerons par une description rapide de l'interface graphique. Puis nous passerons à une présentation sommaire des classes importantes

du Classeur. Cette présentation mettra l'accent sur les décisions de traitement et sur les interactions entre classes.

5.2 Décisions

La discussion qui précède reflète déjà des décisions au niveau de l'énumération des usages et du découpage par classes. Par contre, toute une gamme d'outils de travail est encore possible; toute une gamme de réalisations peut être envisagée. Il faut maintenant examiner les choix possibles, évaluer les possibilités et les conséquences et prendre enfin des décisions. Du côté des décisions visibles pour l'usager, il y a surtout le type d'interface usager et le mode d'accès aux documents HTML; les autres décisions à prendre sont plus transparentes: le langage et les librairies. Finalement, une fois choisi Java, il fallait opter entre une Applet et une Application et puis décider comment construire l'interface.

5.2.1 Java ou C++?

Le prototype est implanté en Java. Bien que j'aie plus d'expérience en programmation C++, les avantages de Java pour un tel projet m'ont semblés assez importants pour justifier la perte de productivité liée à l'apprentissage d'un nouveau langage. *Les principaux avantages de Java pour ce projet étaient: le support direct des URL et de l'accès Internet, sans avoir à passer par une librairie de classes externes; le support multi-plateforme, permettant de transporter le code entre des machines Windows et des machines UNIX sans problème; finalement, le Abstract Window Toolkit (AWT) et le support pour l'animation.* En fait, le AWT représente plusieurs désavantages mais il a suffi aux besoins du prototype. Il faut aussi mentionner la vitalité de la communauté Java et le foisonnement des sources d'information sur le langage.

5.2.2 Applet ou application?

Après une expérimentation initiale où le Classeur était une Applet Java, nous avons trouvé les contraintes trop sévères et l'avons transformé en Application. Avec une Application, le programmeur Java a beaucoup plus de contrôle sur l'environnement de travail de l'usager, mais il perd toute l'infrastructure qui rend si aisée l'écriture des premiers programmes graphiques en Java. S'il veut quand même utiliser le

AWT, directement ou indirectement, il doit reconstruire lui-même quelques classes clef: la fenêtre de l'application, et surtout les deux classes qui se substituent au fureteur quand on lance une application Java graphique: *AppletStub* et *AppletContext*. Plus précisément, le *Classeur* est une classe dérivée de Applet, mais qui ne tourne pas dans un fureteur.

5.2.3 Interactivité?

Ici, les décisions ont été tortueuses. On voulait évidemment un prototype interactif avec une composante graphique intéressante et des éléments de visualisation d'information, si possible. Mais une très grande part de la fonctionnalité du Classeur réside dans la possibilité d'utilisation semi-autonome, en arrière-plan, à des heures où le réseau est moins chargé. Comment faire? Après quelques expériences, je suis parvenue à la solution naturelle: les classes qui constituent le coeur du traitement, comme par exemple la *Collection* et les *Requêtes*, sont des classes mode texte; le même code est exercé ou bien par l'IU ou bien par les outils mode texte bâtis avec des classes comme *l'Enrichisseur* ou enfin à travers le main() de ces classes mode texte principales.

5.2.4 Présentation des documents HTML

Il est important de permettre à l'usager d'examiner les documents repérés: comment, autrement, peut-il juger de leur intérêt et de la pertinence du classement opéré par le système? D'autre part, le *Classeur* - qui était au début une Applet invoquée par un fureteur comme Netscape - ne jouit plus des mêmes privilèges d'affichage de documents HTML depuis qu'il se gère lui-même. Par contre, les droits d'écriture de fichier sont tellement importants qu'ils justifient bien la complexité supplémentaire. Comment afficher des documents HTML? Une solution, rejetée d'emblée pour le manque d'élégance des résultats, consistait en l'affichage du texte des documents dans une fenêtre textuelle. Une autre solution était de construire un éditeur HTML léger, en utilisant les classes Swing par exemple. Une autre possibilité était de construire une Applet qui communiquerait avec le Classeur et qui afficherait les documents. Finalement, *la solution la plus simple était de prendre pour acquis l'existence d'un outil externe tel que Netscape ou Internet Explorer pour visionner les documents; on invoquerait cet outil au besoin. C'est une approche non-intégrée, au sens où le fureteur ne communique pas avec le Classeur, il se contente d'être activé par ce dernier. Toutefois, pour l'usager, cette lacune n'est pas apparente.*

5.2.5 AWT, Swing ou un générateur d'interface?

Dernière décision épineuse: comment produire l'interface? J'ai commencé le développement avec Microsoft J++, qui permet de produire du code d'interface Java pour une IU assemblée graphiquement avec des éléments de l'interface de Windows. Toutefois, je n'avais pas encore exercé cette possibilité quand des considérations de portabilité m'ont convaincue de travailler plutôt dans un environnement Java "pur". À ce point, j'avais encore le choix entre le recours à Swing - le surnom de la boîte à outils d'interface Java, les Java Foundation Classes - ou l'usage de AWT simplement. Swing s'est avéré trop compliqué d'installation, de compilation et de configuration pour les besoins d'un prototype: je me suis donc bornée à l'usage de AWT. (Java est disponible sur le site Internet de Sun, à java.sun.com)

5.2.6 Discussion

Les décisions mentionnées ici ont été prises de façon réfléchie mais elles ont toutes une part d'arbitraire; chacune a eu un impact considérable sur le développement du Classeur. La seule décision dont je ne suis pas satisfaite est celle de développer directement avec AWT: *je pense qu'il est difficile, voire impossible de développer une interface sophistiquée, robuste et polie avec AWT.* Par contre, je refuse de perdre les avantages du multi-plateforme en me cantonnant dans des outils optimisés pour un environnement donné. Le problème est complexe et à la base, il y a la question des standards d'interface: nous n'avons pas de palette d'outils universelle en termes d'aspect et de comportement. Peut-être serait-ce prématuré, les IU, créées à Xerox Parc, n'ayant que 20 ans.

5.3 L'interface graphique

L'interface graphique présente à l'usager une visualisation de sa structure de concepts avec les documents attachés. Elle permet d'indexer la collection interactivement ainsi que de lancer des requêtes et de classer un à un les nouveaux documents obtenus dans la structure existante. Elle permet aussi de modifier le classement apporté par le système. Finalement, elle permet d'examiner des documents HTML en invoquant un navigateur déjà présent dans l'environnement de travail de l'usager. Il y a donc une fenêtre

principale, avec deux zones bien distinctes, l'une textuelle appelée le Panneau Collection et l'autre graphique, appelée le *Panneau* Réseau (Voir figure 5-6, page 62).

5.3.1 Les fenêtres - Réseau et Collection

La fenêtre Réseau permet d'afficher un ensemble quelconque de noeuds et de liens. L'usager peut figer et ajuster la mise en page interactivement. Dans le contexte du Classeur, les graphes visualisés sont, au démarrage, les concepts du curriculum avec leurs sous-concepts. Ces concepts forment les noeuds du graphe et sont visualisés au moyen d'ellipses. Si l'usager charge l'index en mémoire, on verra aussi, au terme de ce processus, des éléments graphiques plus petits et rectangulaires représentant les documents de la collection. La classe *PanneauReseau* est dérivée de la classe *Panel* de Java et supporte l'animation avec d'une *Image* d'arrière-plan. La mise en page est aléatoire et se modifie constamment pour tenter d'atteindre un état d'équilibre où la distance entre deux noeuds liés converge vers une distance cible (voir Modèles à base de ressorts). La classe *PanneauRéseau* est basée sur les classes du répertoire GraphLayout des démonstrations disribuées avec Java mais nous avons fait beaucoup évoluer ce code pour supporter la fonctionnalité dont nous avions besoin.

La fenêtre Collection reflète l'état d'un curriculum et des nouveaux documents repérés mais non encore classés dans ce curriculum. La classe est construite de manière à supporter plusieurs collections mais il y en a toujours une qui est la collection courante et les listes du *PanneauCollection* peignent à l'usager l'état de cette collection. La principale raison d'être de cette classe est d'encapsuler l'accès graphique à la *Collection*, mais elle sert aussi à offrir une vue plus textuelle ainsi qu'à permettre la sélection d'un document ou d'un concept pour diverses opérations offertes à l'usager. Ceci est nécessaire car bien que la sélection graphique des objets, à travers le *PanneauRéseau*, soit possible, elle ne sert pour le moment qu'à des opérations de mise en page.

5.3.2 Menus et fonctions

Nous passons à une description de l'IU du point de vue des commandes offertes à l'usager. Plusieurs de ces commandes correspondent directement à l'un des cas d'usage énumérés en 5.1.1.

Menu Collection

- Construire: l'usager invoque cette commande pour charger en mémoire les structures d'indexation manuelle; si ces structures n'existent pas au moment de l'invocation, elles sont construites dynamiquement (CU4).
- Sauvegarder: enregistre les changements apportés à la Collection depuis le début de la séance ou depuis la dernière sauvegarde. Ceci inclut le rattachement des nouveaux documents à un concept confirmé par l'usager.
- Imprimer: l'usager peut obtenir une copie-papier du réseau de concepts et de documents tel qu'il apparaît couramment dans la fenêtre graphique du panneau réseau.
- Ouvrir: au démarrage, le Classeur actif ouvre par défaut la Collection spécifiée dans sa configuration. Mais l'usager peut déjà couramment afficher un réseau quelconque en utilisant la commande Ouvrir.

Menu Concept

- Décrire: cette commande permet d'examiner les mots-clefs couramment attachés à un concept donné; elle fait surgir un dialogue supportant l'édition de ces mots-clefs (CU6).
- Enrichir: une des commandes principales du Classeur, Enrichir sert à lancer une requête pour un concept donné; au terme de cette opération, une liste des nouveaux documents repérés est disponible pour classement ou examen (CU1).

Menu Document

- Classer: l'usager invoque cette commande pour intégrer un document dans sa structure de concepts. Le système indique le résultat par une nouvelle icône dans la fenêtre graphique (CU2).
- Examiner: cette commande permet d'examiner un document HTML, peu importe sa provenance ou son adresse physique. Elle active pour ce faire un fureteur déjà présent dans l'environnement de l'usager (CU5).
- Ajuster: cette commande sert à confirmer un classement ou à le modifier. L'usager obtiendra une confirmation graphique du déplacement du document dans l'arbre des concepts. Et une rétroaction sera effectuée (CU3).

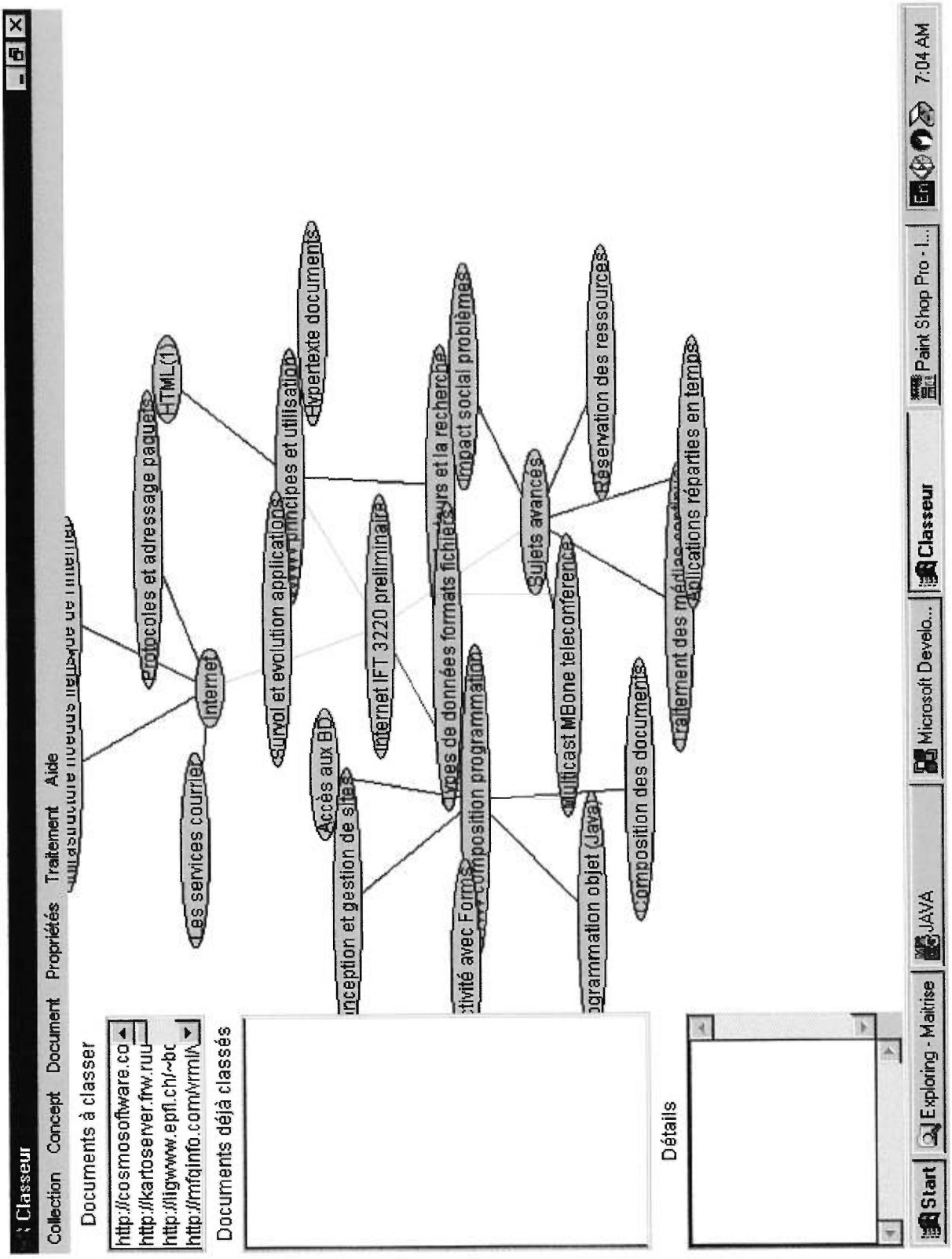

Figure 5-6: Le classeur au démarrage, avant l'intervention de l'usager

Menu Propriétés

- Configurer: les principaux paramètres d'opération du Classeur actif peuvent être modifiés à travers cette commande; on peut ainsi changer le SRI par défaut, le nom des fichiers de support et ainsi de suite.

Menu Traitement

- Arrêter: si l'usager se fatigue de l'animation dans le panneau réseau, il peut la suspendre au moyen de cette commande; cette animation suit les étapes de l'affichage automatique des noeuds du réseau.
- Continuer: l'usager peut permettre à la mise en page automatique de continuer à se stabiliser en invoquant la commande Continuer, qui alterne dans le menu avec la commande Arrêter.

Menu Aide

- Expliquer: cette commande résulte en l'affichage de l'aide du Classeur actif dans une fenêtre de furetage. Cette facilité est encore peu développée, à vrai dire.

5.3.3 Commentaires sur l'interface graphique

Mis à part le Panneau Collection, qui est sans doute la partie la moins satisfaisante de cette interface, nous avons dans le Classeur une façon adéquate de faire les entrées de données nécessaires, de configurer les éléments qui s'y prêtent ainsi que de visualiser les résultats des traitements. Le Panneau Réseau, bien qu'encore primitif, résout le problème de mise en page d'un réseau de données par le biais d'une méthode aléatoire et itérative (voir section 2.3), tout en se prêtant aussi à un positionnement direct des noeuds par l'usager [Fric99]. La visualisation rend bien compte de la structure de concepts ainsi que du nombre de documents attachés à chaque concept. Il serait bon d'avoir plus de capacité d'interaction graphique avec la collection (voir chapitre VII).

Nous terminons la description du prototype en évoquant les classes qui correspondent à l'univers du problème, c'est-à-dire la *Collection*, le *NoeudConcept* et les classes de *Requête*.

5.4 La classe Collection

La *Collection* représente le corpus de documents. Elle donne accès aux fonctions d'indexation, de comparaison et de classement. C'est une composante non-graphique. Elle peut être invoquée à partir de la ligne de commande ou à travers le Classeur. *La Collection s'articule autour des concepts et des documents rattachés à ces concepts. Elle représente l'un des univers conceptuels de l'usager; en particulier, pour un concepteur pédagogique, un professeur ou un étudiant, elle représente un curriculum.* Plus précisément, au niveau des objets, une *Collection* regroupe une structure de concepts, des documents, une index automatique et un index manuel.

5.4.1 L'index automatique

L'index automatique est constitué par deux structures coordonnées par la *Collection*. Les termes d'indexation forment un tableau où sont consignés, pour chaque terme d'indexation, son nombre d'instances pour chaque concept ainsi que diverses autres données, primitives ou dérivées d'une indexation automatique des documents de la collection. Avec les collections de test utilisées jusqu'ici, on atteint les 7.000 termes d'indexation, mais on peut envisager facilement 10.000 ou 12.000 termes. Les comparaisons de chaînes de caractères sont une opération coûteuse dans la plupart des langages de programmation. En Java, pour comparer plus de dix chaînes, on recommande une table de hachage. [Ball99] Pour éviter des comparaisons de chaînes à répétition, les termes d'indexation sont installés, au fur et à mesure de leur découverte, dans une table de hachage qui permet de récupérer leur adresse dans la matrice des termes d'indexation. Ceci constitue la deuxième structure coordonnée pour l'index libre, la première étant le fichier inversé implanté comme un tableau d'objets de la classe *NoeudIndex*.

Notons que le lexicaliseur rejette les termes contenus dans une liste négative (stoplist) et accepte les termes contenus dans une liste positive. La liste négative est l'une des listes documentées dans la littérature des SRI [Fox92]. La liste positive est fournie par l'usager. Les deux listes, positive et négative, sont fournies par le prototype sous la forme d'un fichier ASCII; il est donc loisible à l'usager de les éditer

à sa guise ou encore d'utiliser ses propres fichiers en les spéficiant à travers la configuration des propriétés du système (Propriétés | Configurer).

5.4.2 L'index manuel

Les termes d'indexation automatique fournissent un bon point de départ pour l'enrichissement des concepts d'un curriculum, à condition que chaque concept possède déjà des documents rattachés, à condition aussi que ces documents soient disponibles pour l'indexation. Or au stade préliminaire de la conception certains concepts ne sont pas encore étayés par des références. De plus, certains documents peuvent ne pas être accessibles au moment de l'indexation de la collection. Pour mieux supporter le classement et l'enrichissement, on fournit donc aussi un index contrôlé, "manuel". Reportons-nous à la figure 5-5. Pour accélérer le processus de développement d'un tel index par l'usager, le prototype crée les termes candidats dans le cadre de son analyse du fichier de curriculum. Il est à noter que cet index contrôlé supporte les lexies complexes (expressions), ce qui est un apport important en regard de l'index automatique. Il serait intéressant de supporter une balise qui permettrait au concepteur de spécifier ces termes d'indexation au niveau du fichier curriculum original.

Le tableau des concepts et le tableau des documents forment, avec l'index, le coeur de la *Collection*. Le tableau des concepts est créé initialement au terme du traitement effectué sur le curriculum normalisé (voir 5.1.3). Nous passons maintenant à la description des classes ayant servi à implanter les objets de type *Concept*. Quant à la classe qui représente les instances des documents, *NoeudDocument*, elle est plutôt pauvre et passive à l'heure actuelle: elle pourrait être enrichie dans un contexte où l'on ferait une analyse textuelle qui irait au-delà des statistiques actuelles.

5.5 La classe NoeudConcept

Les éléments du curriculum sont supportés par deux classes: le *NoeudConcept* et le *FichierHTML*. Un *NoeudConcept* est l'objet de base qui représente un concept dans notre prototype. *Notons que la représentation ne se prête pas au raisonnement, comme en IA, mais qu'elle vise seulement la présentation d'une structure relationnelle où l'on puisse épingler des termes d'indexation manuelle avec leur poids, d'autres concepts et des documents.* Un *FichierHTML*, dans le contexte des concepts, sert à analyser un curriculum rédigé par un usager. Le nom du concept, une chaîne de caractères contenant un ou plusieurs mots, est construit durant l'analyse du curriculum. Le nom est relativement court, pour

alléger l'affichage: l'usager peut le modifier à sa guise pour corriger un nom synthétique jugé inapproprié. Le *NoeudConcept* entrepose aussi les noms de tous les concepts avec lesquels un concept entretient des liens, en amont ou en aval.

5.5.1 Les mots-clefs d'un concept

Le tableau des mots-clefs d'un concept est construit au cours du processus d'analyse du curriculum. Ce tableau préliminaire sert seulement d'aide au démarrage pour l'usager, qui devrait l'épurer pour n'en conserver que les termes les plus significatifs. Ces termes peuvent comporter un ou plusieurs mots, et on leur assigne initialement un poids de un. Un poids non-unitaire pour un terme d'indexation dénote une modification, soit manuelle, soit due à la rétroaction de pertinence. Il s'agit avec les mots-clefs d'améliorer la qualité du classement.

5.6 Les classes de Requête

Une requête créée à travers l'interface graphique se résume essentiellement à une *RequêteClasseur*. L'émission d'une requête crée un objet *RequêteClasseur* qui prend comme point de départ un concept décrit par une étiquette et un certain nombre de mots-clefs. Elle transforme ces données et les combine avec le choix d'un SRI sur Internet pour former une requête physique, la transmettre au SRI, recueillir les résultats et procéder à un tri élémentaire. La requête est créée à travers l'interface graphique - menu Concept, item Enrichir - ou encore au moyen de la ligne de commande; l'un de ses paramètres est une chaîne de caractères composée de plusieurs termes. La chaîne est lue et la requête est dépouillée des mots non significatifs. La classe *RequêtePhysique* fait tout le travail d'établissement de la communication avec un SRI, de lancement de la requête avec la syntaxe appropriée pour le SRI en question, d'attente et de traitement de la réponse fournie par le SRI. En fait, puisque le prototype est écrit en Java, c'est une tâche facilitée par des classes comme *URL* et *URLConnection*.

5.7 Conclusion

L'objectif de ce chapitre était de présenter le prototype que nous avons construit pour illustrer notre approche au problème de l'enrichissement automatique d'un réseau de concepts. Nous avons reparlé des mécanismes principaux, mais cette fois en termes d'implantation. Nous avons évoqué l'architecture, les

décisions, les grandes classes et l'interface graphique. Nous décrirons maintenant comment le Classeur supporte la tâche de l'usager, en détaillant les interactions mais surtout en examinant les résultats de ces interactions. Nous tenterons de voir si le Classeur répond à nos attentes.

Chapitre VI: Expérience pratique avec le Classeur

Dans quelle mesure le prototype répond-il à nos attentes en ce qui a trait à la qualité de l'enrichissement et à la qualité du classement? Arrive-t-on à formuler automatiquement des requêtes raisonnables pour chaque concept? Est-ce qu'on récupère effectivement des documents pertinents ou bien si l'on se heurte à des problèmes d'homonymie difficiles à régler sans thesaurus? L'extraction automatique par le prototype de termes candidats pour l'indexation manuelle est-elle un avantage ou bien nuit-elle au travail? La rétroaction améliore-t-elle vraiment la précision du classement? C'est ce que nous tenterons d'élucider par quelques expériences.

Notons que le Classeur *apprend à classer* de deux façons: quand un document est ajouté à la collection pour un concept donné, il change la représentation du concept au moyen des termes d'indexation automatique. De plus, quand l'usager offre une rétroaction au système, la représentation du concept cible jugé correct par l'usager change au niveau des termes d'indexation manuelle. Nous mènerons donc deux séries d'expériences avec le prototype: dans la première série, la collection restera intacte et chaque opération du système sera sans lendemain; dans la deuxième série, le Classeur pourra apprendre, dans une certaine mesure, à émuler les jugements de l'usager, en améliorant ses poids.

Reportons-nous un moment à la figure 6-1, décrivant le flot des données dans le Classeur. Il y a beaucoup d'éléments hors de notre contrôle dans ce système: les SRI externes, les collections externes, ainsi que la collection interne, le curriculum original, sont des données hors de notre zone d'influence. On peut bien sûr choisir un SRI parmi d'autres; on peut aussi exclure certaines collections externes, certains sites, mais ceci se fait au niveau du post-traitement; on pourrait aussi privilégier certaines collections externes, mais il faudrait pour cela utiliser des options disponibles avec certains SRI et pas avec d'autres, nous avons écarté cette possibilité pour l'instant. Il faudra se contenter d'exploiter le mieux possible l'information présente dans le curriculum original, et faire évoluer cette information avec le temps.

On peut envisager le projet sous l'angle suivant; étant donné une collection interne, des SRI externes et des collections externes, peut-on construire des mécanismes qui vont permettre à la collection interne de s'enrichir, de façon structurée et semi-autonome, avec le temps? Quelles sont les limites de cette approche? Obtient-on ainsi un bon assistant? Pourrait-on éventuellement obtenir un meilleur assistant? (Voir figure 6-1).

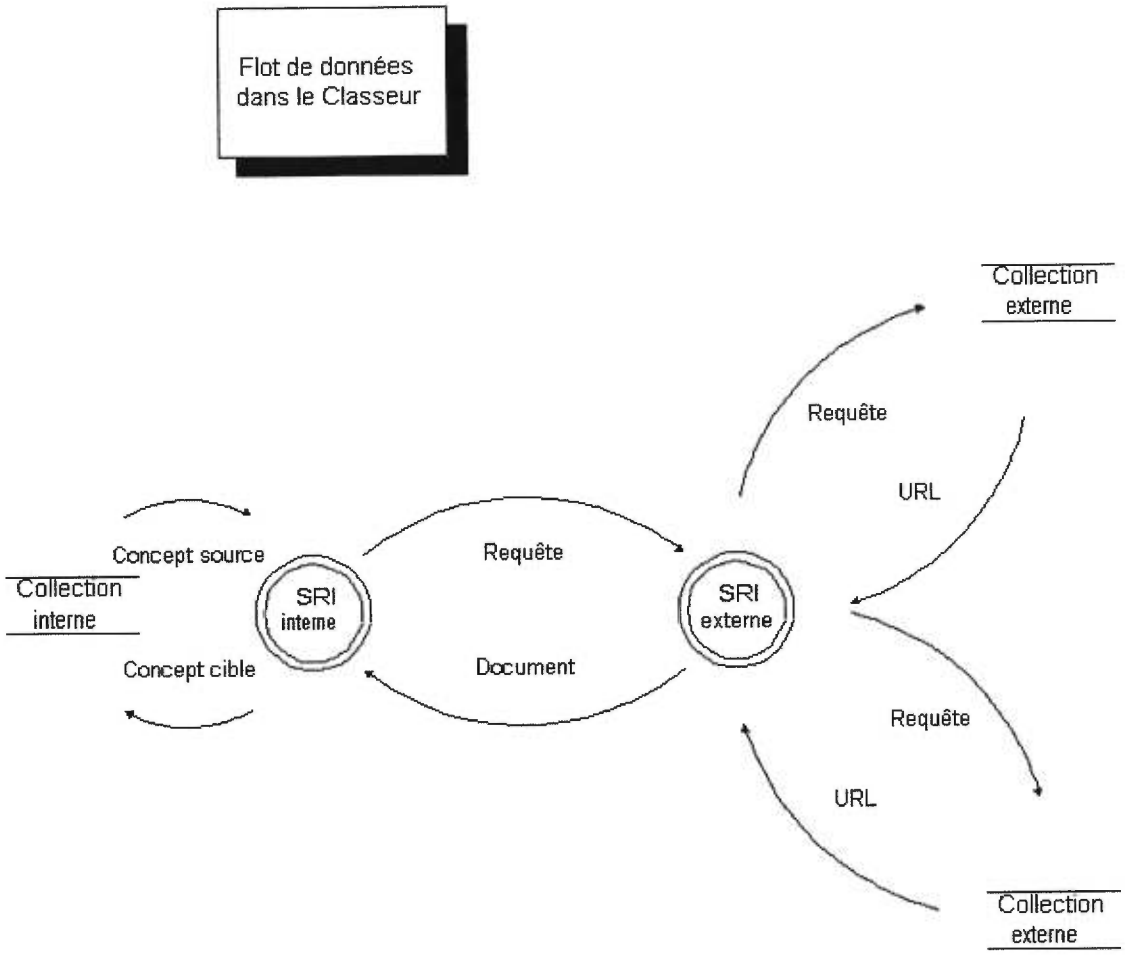

Figure 6-1: Flot des données dans le Classeur actif

Nous tentons de résoudre ces questions dans ce chapitre et le suivant. Nous commençons par décrire les requêtes et le classement en mode interactif; cette description formera la section un. Puis, nous oublions les aspects interactifs pour nous concentrer sur l'aspect global, mieux desservi par les outils de commandes. Pour fins d'évaluation, nous avons isolé le mieux possible les divers chemins de traitement dans la figure ci-dessus. Par conséquent, à la section deux, il s'agit d'évaluer la qualité du flot qui part d'un concept, formule une requête, la lance et voit finalement le SRI externe retourner un document jugé pertinent. Pour isoler la fonction requête de la fonction classement, nous avons jugé nous-même du classement approprié.

Nous procédons de la même manière pour le classement, c'est-à-dire que nous faisons alors abstraction de l'aspect requête; pour revenir à la figure 6-1, nous prenons un document en supposant qu'il nous a été retourné par un SRI externe et nous évaluons la justesse du classement apporté par le Classeur. Cette

analyse fait l'objet de la section trois. Nous discutons ensuite les problèmes rencontrés: la section quatre porte sur les requêtes et la section cinq sur le classement. La section six décrit le cycle de vie d'un curriculum avec le Classeur actif. Ceci nous conduira au chapitre VII, où nous replacerons le Classeur dans un contexte de gestion d'information sur Internet.

6.1 Le Classeur en mode interactif

Notre propos dans cette section est de montrer comment l'usager se sert du Classeur actif graphique pour son travail d'enrichissement. Nous décrirons le mode d'emploi et ajouterons des saisies d'écran pour illustrer le fonctionnement. Nous parlerons du lancement d'une opération d'enrichissement, de la façon dont l'usager est averti des résultats; puis nous exposerons les opérations relatives au classement: demande de classement, examen du document, demande d'ajustement du classement. Finalement, nous montrerons qu'il est possible de changer les mots-clefs à travers l'interface usager.

6.1.1 Lancement d'une requête

Pour lancer une requête, et parce que nos requêtes visent toujours à enrichir un concept donné, l'usager choisit la commande Enrichir du menu Concept. (Notons que les requêtes peuvent être lancées sans que l'index de la collection soit chargé en mémoire: la saisie d'écran ne montre actuellement que les concepts, pas les documents.) Le Classeur présente alors à l'usager une liste des concepts dans le curriculum courant; quand il choisit l'un de ces concepts, la liste des mots-clefs est affichée dans une zone éditable de la boîte de dialogue. L'usager peut ajouter ou retrancher des termes à ce niveau, sans effet permanent sur la structure. Lorsqu'il active ensuite le bouton Traiter, une connection Internet est établie au besoin et une requête est envoyée au moteur de RI spécifié dans ses options de configuration.

6.1.2 Résultats de la requête

L'émission d'une requête se termine normalement au moment où la liste d'URL repérés par le moteur de RI externe, après avoir fait l'objet d'un filtrage préliminaire, est retournée à l'usager. *L'usager reçoit un message dans la zone de messagerie située au bas de l'écran; si tout va bien, ce message lui indique le nombre d'URL retenus; autrement, le message indique le type de problème rencontré: impossibilité*

d'accès Internet, problème de connection avec le moteur de RI ou encore manque de documents pertinents pour la requête donnée. La liste de ces URL sera maintenant affichée dans la zone Nouveaux Documents, dans la partie gauche de l'écran.

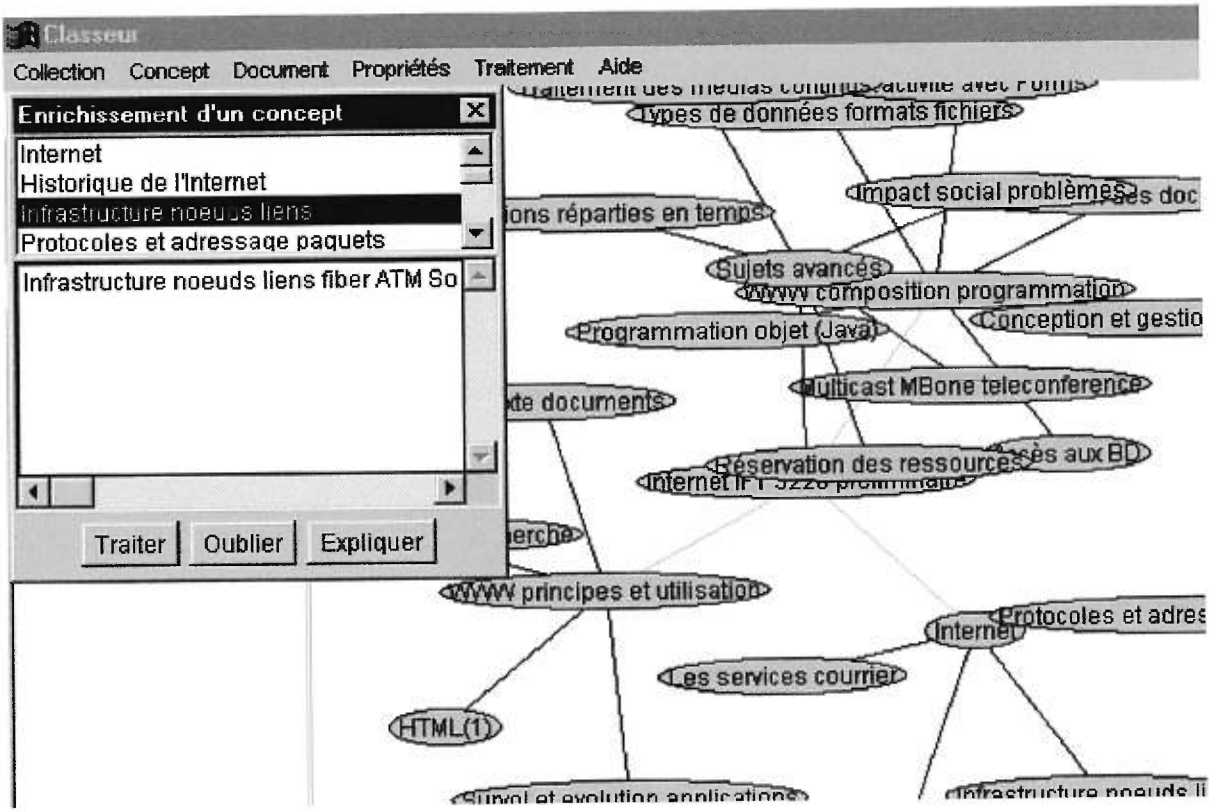

Figure 6-2: Lancement d'une requête en mode interactif

6.1.3 Demande de classement d'un document

Le Classeur organise les documents en les comparant aux documents déjà rattachés et en s'aidant des mots-clefs. Il faut donc que la collection soit déjà indexée en mémoire, une opération effectuée en choisissant la commande Construire du menu Collection. Par la suite, l'usager classe un document en le sélectionnant dans une liste ou encore en spécifiant l'URL manuellement. Le prototype procède alors à l'indexation du document choisi et compare les résultats à la description produite au préalable pour chacun des concepts de la collection. Le concept retenu est affiché dans la zone de messagerie et le

document est maintenant représenté par une grande icône jaune et rattaché à ce concept dans la zone graphique. Les autres documents figurent comme de petites icônes bleues anonymes.

Notons que la saisie d'écran à la figure 6-3, plus bas, reflète l'état du Classeur avant l'édition des mots-clefs: les concepts zéro et un y possèdent encore une importance indue. (Rappelons que le numéro de concept est produit par le mécanisme décrit en 5.1.3. Le concept 0, "Internet IFT 3220 Préliminaire", se situe en fait hors du curriculum et servirait éventuellement à intégrer divers cours dans une même structure. Son nom a été produit à partir de la balise <Title> du document. Par contre le concept 1, "Internet", est le tronc de l'arbre représentant le curriculum). Nous corrigerons cette situation de mauvaise classification tel que décrit au paragraphe 6.3.3 ci-dessous.

6.1.4 Examen interactif d'un document

Pour ne pas trop interrompre la continuité de l'environnement de travail, nous voulions permettre à l'usager d'examiner un document sans sortir du Classeur pour lancer son navigateur; par contre, nous n'avons pas codé un navigateur intégré, donc nous nous contenterons de lancer le navigateur à la place de l'usager. Ceci possède l'avantage de pouvoir intégrer la recherche du document en question à travers les chemins de documents spécifiés par l'usager comme option de configuration. Cette fonction d'examen d'un document est disponible par la commande Examiner du menu Document. Il va sans dire que les documents sur l'Internet sont examinés tout aussi bien que les documents locaux.

6.1.5 Ajustement du classement par l'usager

Après avoir examiné le document pour juger lui-même de son thème principal, l'usager voudra confirmer ou modifier le classement apporté par le Classeur actif. *Si l'usager ne procède pas à cet ajustement ou à cette confirmation explicite, le document n'est pas intégré au curriculum.* L'usager ajuste le classement en choisissant la commande Ajuster du menu Document. On lui présente alors une liste des concepts disponibles pour y rattacher le document. Éventuellement - mais ce n'est pas encore possible à l'heure actuelle - il pourra choisir de créer un concept pour y ajouter ce document. Quand l'usager active le bouton Traiter, le Classeur retire le document de son point d'attache temporaire pour le redessiner à sa place définitive dans la structure de concepts et de documents.

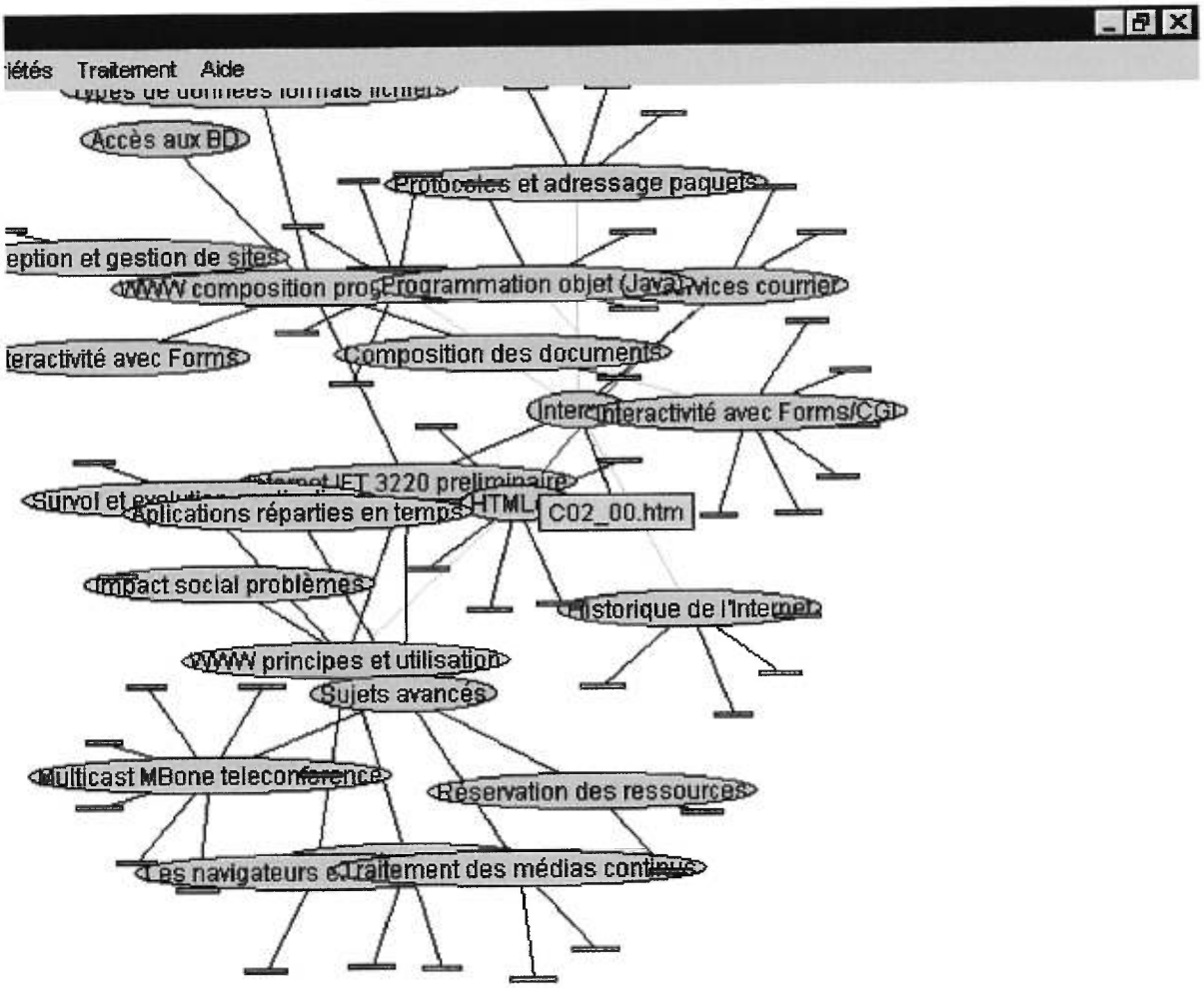

Figure 6-3: Après une demande de classement pour C02_00.htm

6.1.6 Édition des mots-clefs décrivant un concept

La dernière opération importante du Classeur en mode graphique est l'édition d'un concept, en particulier pour ce qui est des mots-clefs. Initialement, les mots-clefs sont extraits automatiquement du fichier curriculum créé par l'usager. Ceci a pour but d'aider l'usager à démarrer dans son édition. Quand l'usager choisit d'ajouter ou de retrancher un mot-clef, il le fait par le biais de la commande Éditer du menu Concept. Le Classeur lui présente alors une liste des concepts; quand l'usager choisit un concept, la liste des mots-clefs est affichée dans une zone éditable; l'usager peut alors ajouter ou retrancher des mots-clefs. Ces derniers seront dès lors utilisés par le système pour améliorer ses classifications et ses requêtes pour le concept en question.

Les zones de fonctionnalité ciblées par le Classeur actif, émission de requêtes et classement semi-autonome de documents dans le curriculum, sont donc bien couvertes par le mode interactif d'opération. Toutefois, la meilleure façon d'évaluer la performance du Classeur reste l'exercice en mode lot. Dans le reste de ce chapitre, nous oublierons donc le Classeur graphique pour nous concentrer sur les invocations à partir de la ligne de commande. Ce mode nous permettra de nous former une meilleure idée des possibilités et des lacunes de l'approche employée dans ce prototype. Nous commencerons par observer la formulation ainsi que le succès ou l'échec des requêtes.

6.2 Évaluation des requêtes

L'une des deux fonctions importantes du prototype est l'émission semi-autonome de requêtes visant à repérer sur l'Internet des documents susceptibles d'enrichir un curriculum. Nous nous concentrerons sur la précision plutôt que le rappel, pour diverses raisons: étant donné que l'univers des documents est dynamique, il est difficile d'évaluer la proportion des documents pertinents qui ont effectivement été retrouvés; mais surtout, le type d'application prévu pour ce Classeur actif est tel qu'il s'agit simplement d'utiliser les documents repérés comme ressources supplémentaires, pour illustrer des concepts déjà relativement bien couverts par des documents locaux. Nous avons exercé le prototype en utilisant le squelette de cours sur Internet exploité pour l'analyse.

6.2.1 Précision du repérage

Étant donné qu'une requête vise à enrichir un concept particulier, l'émission d'une requête peut se terminer de quatre façons:

- *aucun document n'est repéré: voir le chemin issu de R0, à la figure 6-4;*
- *un document repéré est bien pertinent pour le concept; c'est le cas du document D0;*
- *un document repéré est pertinent pour un autre concept du réseau; par exemple, le document D1;*
- *finalement un document n'est absolument pas pertinent dans le contexte du réseau de concepts en question; par exemple, D2.*

Nous ignorerons pour l'instant le cas où aucun document n'est repéré.

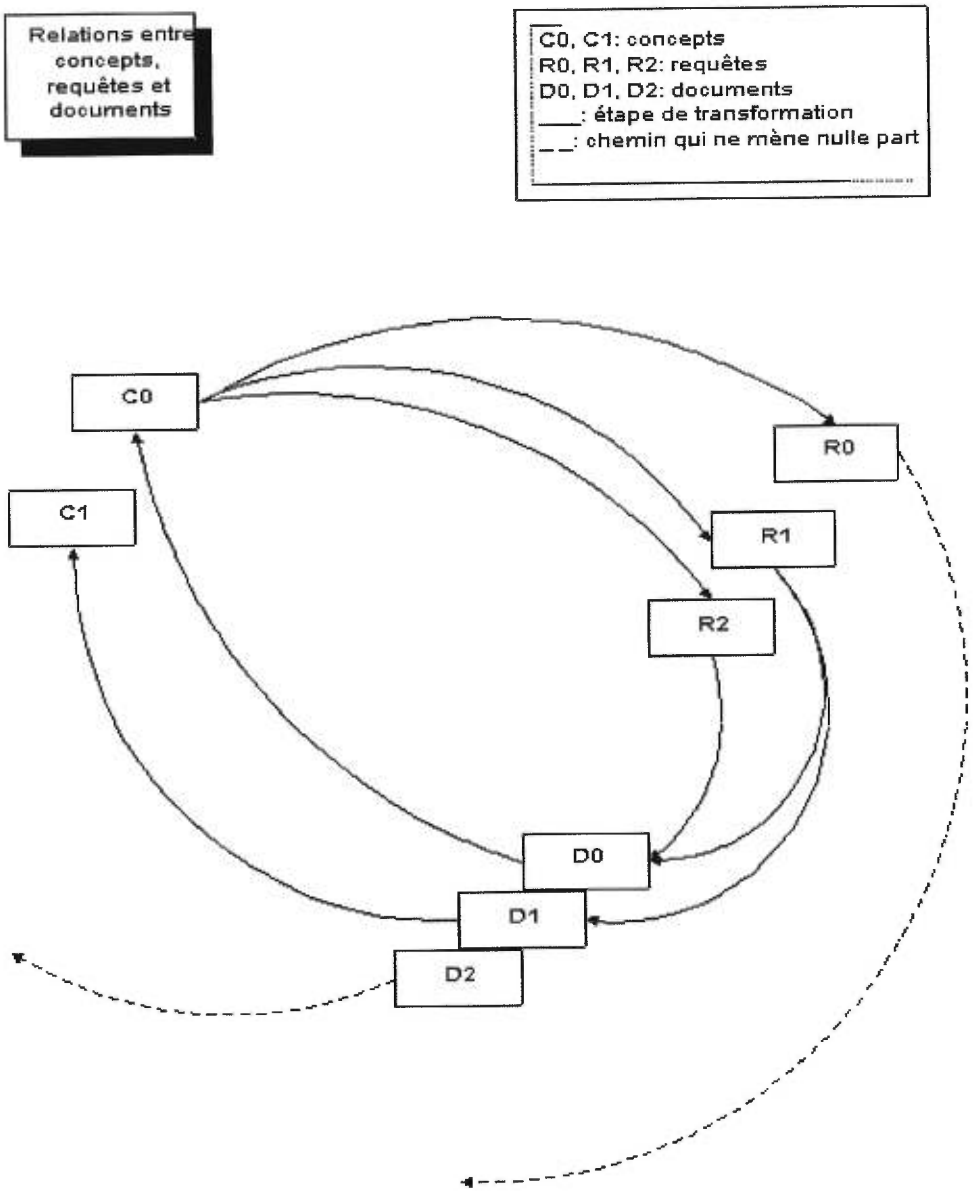

Figure 6-4: Relations entre concepts, requêtes et documents

6.2.2 Expérimentation

La classe *RequêteClasseur*, utilisée en mode interactif, n'est pas assez puissante pour l'expérimentation en mode lot. C'est pourquoi nous avons construit une autre classe, décrite déjà au chapitre V, et appelée *RequêteConcepts*. Cette dernière peut s'adresser à plusieurs moteurs de RI au cours de la même invocation et elle compte les ressources potentiellement pertinentes qu'elle a pu récupérer, jusqu'à ce que son objectif soit atteint. C'est donc cette dernière classe que nous avons utilisé pour évaluer les requêtes.

L'expérimentation consiste en l'exercice de la classe *RequeteConcepts* dans une boucle: pour chaque concept de la collection et pour chaque longueur de requête (deux termes, pour le moment) on crée une *RequeteConcepts* et on lui demande de repérer vingt documents pour le concept en question. Un rapport global est produit sous la forme d'un fichier HTML; ce rapport contient, pour chaque document repéré, un lien hypertexte permettant d'examiner le document pour juger manuellement de sa pertinence pour le concept cible.

6.2.3 Résultats des requêtes

Il s'agit donc d'évaluer la qualité des requêtes sur la base de leur dénouement. Pour chaque concept, nous émettons des requêtes à chaque engin de RI. Pour chaque opération, nous notons, parmi les documents les mieux cotés par les engins de RI, la proportion tombant dans chacune des catégories mentionnées plus haut. Notons que pour évaluer le repérage de façon indépendante du classement, nous procédons au classement manuellement. Étant donné que l'évaluation manuelle est très longue, nous avons expérimenté avec des requêtes à deux termes seulement. Il faut noter que c'est justement parce que l'évaluation manuelle est si longue qu'il vaut la peine d'automatiser la tâche.

Dans le tableau III, les résultats des requêtes sont groupés selon le concept pour lequel la requête a été fabriquée (rangées); la première colonne indique le numéro de concept, la deuxième colonne les termes de la requête; la troisième colonne indique la proportion des documents obtenus qui portaient effectivement sur le concept en question; la quatrième colonne indique la proportion qui portaient sur un autre concept du même curriculum; la cinquième colonne indique la proportion de documents sans rapport avec le curriculum et la dernière colonne consigne des observations sur quelques résultats. Les moteurs utilisés étaient AltaVista, Inference, Lycos et Profusion.

Concept	**Requête**	**Concept Cible**	**Autre Concept**	**Erroné**	**Commentaires**
1	Internet	Profusion: 0.1	Profusion: 0.7	Profusion: 0.2	
2	Historique Internet	Profusion: 0.0	Profusion: 0.2	Profusion: 0.8	Pauvre comme résultat. Voir tableau IV, obtenu pour même requête, lors d'une autre séance.
3	Infrastructure noeuds	Inference: 0.1 Profusion: 0.1	Inference: 0.1 Profusion: 0.1	Inference: 0.8 Profusion: 0.8	Problème d'homonymie: la plupart des

					documents portaient sur les "noeuds" marins.
4	Protocoles adressage	Inference: 0.5 Profusion: 0.3	Inference: 0.0 Profusion: 0.3	Inference: 0.5 Profusion: 0.4	L'un des termes, "protocoles" est d'application beaucoup trop générale.
5	Services courrier	Altavista: 0.0 Inference: 0.7	Altavista: 0.0 Inference: 0.0	Altavista: 1.0 Inference: 0.3	
6	WWW principes	Altavista: 0.2 Inference: 0.0	Altavista: 0.5 Inference: 0.3	Altavista: 0.3 Inference: 0.7	La plupart des résultats portaient sur la scientologie ou l' impact social de l'Internet.
7	Hypertexte documents	Altavista: 0.3 Inference: 0.2 Profusion: 0.8	Altavista: 0.3 Inference: 0.6 Profusion: 0.2	Altavista: 0.4 Inference: 0.2 Profusion: 0.0	Le terme "Documents" est beaucoup trop général.
8	HTML(1)	Lycos: 0.6 Profusion: 0.4	Lycos: 0.0 Profusion: 0.3	Lycos: 0.4 Profusion: 0.3	
9	Navigateurs recherche	Profusion: 0.1	Profusion: 0.1	Profusion: 0.8	La plupart des résultats portaient sur les "Navigateurs" humains.
10	Survol évolution	Altavista: 0.0 Inference: 0.0 Lycos: 0.0	Altavista: 0.0 Inference: 0.0 Lycos: 0.0	Altavista: 1.0 Inference: 1.0 Lycos: 1.0	La plupart des documents portaient sur la théorie de l'évolution. Les pîres résultats.
11	WWW composition	Profusion: 0.4	Profusion: 0.0	Profusion: 0.6	
12	Composition documents	Profusion: 0.3	Profusion: 0.0	Profusion: 0.7	Les termes sont beaucoup trop généraux.
13	Types données	Inference: 0.4 Profusion: 0.1	Inference: 0.2 Profusion: 0.3	Inference: 0.4 Profusion: 0.6	
14	Interactivité Forms	Lycos: 0.2 Profusion: 0.4	Lycos: 0.4 Profusion: 0.0	Lycos: 0.4 Profusion: 0.6	
15	Accès BD	Lycos: 0.1 Profusion: 0.0	Lycos: 0.0 Profusion: 0.0	Lycos: 0.9 Profusion: 1.0	BD refère plutôt aux Bandes Dessinées qu'aux Banques de

					Données.
16	Programmation objet	Altavista: 0.6 Profusion: 0.5	Altavista: 0.1 Profusion: 0.1	Altavista: 0.3 Profusion: 0.4	
17	Conception gestion	Altavista: 0.4 Lycos: 0.1 Profusion: 0.3	Altavista: 0.0 Lycos: 0.1 Profusion: 0.1	Altavista: 0.6 Lycos: 0.8 Profusion: 0.6	Chanceux, ne marcherait pas pour un autre curriculum: les termes sont vagues.
18	Sujets avancés	Altavista: 0.0 Profusion: 0.0	Altavista: 0.6 Profusion: 0.4	Altavista: 0.4 Profusion: 0.6	Chanceux, ne marcherait pas pour un autre curriculum.
19	Traitement médias	Profusion: 0.2	Profusion: 0.0	Profusion: 0.8	
20	Multicast MBone	Profusion: 0.8	Profusion: 0.0	Profusion: 0.2	Ce sont les meilleurs résultats: termes très spécifiques.
21	Réservation ressources	Altavista: 0.0 Inference: 0.0 Profusion: 0.0	Altavista: 0.0 Inference: 0.3 Profusion: 0.0	Altavista: 1.0 Inference: 0.7 Profusion: 1.0	Termes trop génériques. La plupart des documents traitaient de réservations d'hôtels.
22	Applications réparties	Inference: 0.1 Lycos: 0.0	Inference: 0.3 Lycos: 0.4	Inference: 0.6 Lycos: 0.6	
23	Impact social	Altavista: 0.1 Inference: 0.0 Lycos: 0.6	Altavista: 0.0 Inference: 0.0 Lycos: 0.2	Altavista: 0.9 Inference: 1.0 Lycos: 0.2	

Tableau III: Évaluation de qualité des requêtes

Pour tenter de résumer ce tableau et obtenir une mesure globale de précision, nous avons tout simplement calculé la moyenne des entrées à la colonne Concept Cible, traitant les résultats obtenus pour des SRI différents comme des données indépendantes. Nous obtenons ainsi une précision de 22.7%. Cette mesure peut sembler basse mais elle est très bonne étant donné la précision des SRI sur Internet (voir 6.7).

Les résultats semblent démontrer que l'approche est raisonnable. *Je me dois de mentionner toutefois un problème lié au classement des documents: le fait de classer un document comme "appartenant" à un concept, par exemple 12, Composition des documents, laisse entier le problème de la catégorisation du*

nouveau document. Il peut s'agir d'un traité de composition comme il peut s'agir d'une page publicitaire. Pour être vraiment utile dans un contexte d'enseignement, le Classeur actif devrait être capable d'une catégorisation plus fine et d'un rejet des objets de type inapproprié. On peut envisager des paramètres configurables par l'usager.

Nous discuterons des résultats des requêtes plus bas, à la section 6.4. Mais auparavant, nous présentons les résultats de nos expériences de classement de documents.

6.3 Évaluation du classement

L'autre fonction importante du prototype est le classement des documents. Étant donné un document à classer, et peu importe d'où provienne ce document, dans quelle mesure le concept choisi par le système pour y rattacher le document est-il le même qu'aurait choisi l'usager? On peut se demander aussi comment évolue dans le temps la précision du classement. Enfin, un autre facteur à explorer sera la façon de combiner la comparaison basée sur les termes d'indexation automatique avec la comparaison basée sur les termes choisis manuellement.

6.3.1 Précision du classement

Nous évaluerons la précision du classement de la façon suivante. *Pour chaque concept, nous choisirons quelques documents pertinents; nous demanderons au système de classer ces documents.* Rappelons qu'un document est classé comme appartenant au concept N si et seulement si le vecteur représentant le concept N est le plus proche du vecteur représentant le document dans l'espace vectoriel décrivant la collection. Rappelons aussi que par le plus proche, nous entendons celui qui maximise la proximité telle que définie par la mesure du cosinus (section 2.1.2). Rappelons enfin qu'il y a deux de ces espaces: celui basé sur l'indexation automatique et celui basé sur l'indexation manuelle.

Nous évaluerons simplement la qualité du classement en termes de la proportion des documents classés correctement. Pour avoir une meilleure idée du traitement, nous tiendrons compte également des cas où le concept jugé deuxième aurait dû être classé premier. Nous ferons d'abord cette évaluation sans ajouter aucun document à la collection et sans offrir de rétroaction au système. Puis nous ajouterons les

documents à la collection et nous offrirons une rétroaction (au niveau du Classeur interactif, les deux opérations sont indissociables mais au niveau des commandes elles sont facilement dissociées).

6.3.2 Expérimentation

Pour évaluer le classement, on prépare d'abord un fichier contenant les noms des documents à classer. Le code d'évaluation analyse le fichier de documents à classer; pour chaque document trouvé dans ce fichier, il demande à la Collection de classer le document en consignant ses résultats intermédiaires dans un Vector passé en entrée au constructeur de Collection. Les étapes sont donc les suivantes:

1. Préparer des structures d'entrée de données supplémentaires où l'objet Collection consignera ses résultats intermédiaires.
2. Lancer le classement et le laisser opérer normalement.
3. Une fois le classement terminé, passer les données accumulées à un objet *EvalClassement*

6.3.3 Classement avec termes d'indexation automatique seulement

Rappelons maintenant comment la similarité entre un concept et un document est calculée (nous en avons parlé à la section 2.1.2). Un modèle de chaque concept est d'abord construit: ces modèles sont produits au terme de l'indexation libre des documents déjà rattachés à ce concept au début de la session. On obtient donc un vecteur pour chaque concept. Pour déterminer à quel concept rattacher un nouveau document, on procède à l'indexation libre de ce document, ce qui produit un autre vecteur, puis on calcule le cosinus de l'angle entre le vecteur document et les vecteurs concepts. Les résultats sont des nombres réels: on les ordonne et le plus grand est retenu comme indiquant le concept pertinent.

Dans la table qui suit, le concept cible est celui retenu comme étant le véritable point d'attache d'un document. Ce jugement peut avoir été porté par le concepteur du cours ou encore par moi au moment de l'expérimentation. Le premier concept est celui retenu par le système, c'est-à-dire celui pour lequel le cosinus était le plus grand. Comme nous avons accès aux calculs intermédiaires, nous savons aussi quel était le prochain concept candidat dans la liste, par ordre de pertinence du document pour le concept. Nous le consignons aussi. Dans la table ci-dessous, les jugements corrects sont en caractères gras.

URL ou Titre	Concept Cible	Premier concept	Second concept
A Brief History of the Internet	2	**2**	7
The History of the Internet	2	**2**	7
History of the Internet and Web	2	**2**	7
Roads and Crossroads of Internet History	2	**2**	7
Internet and World Wide Web History	2	**2**	19
Internet History	2	**2**	7
Internet Society ... History of the Internet	2	**2**	19
History of the Internet and the WWW	2	7	**2**
Yahoo! Computers and Internet: History	2	**2**	19
NetHistory	2	**2**	19
... the Speed ... comparison of data rates	3	**3**	2
Introduction au réseau Internet	4	9	2
ISPs.com -- High Speed Modems	3	**3**	7
Organisation de l'Internet	3	**3**	2
Survol historique -- Modèle client serveur	3	9	**3**
Reducing WWW Latency ...	21	**21**	19
The Rodeo Group	21	19	2
Organisation matérielle du réseau	3	2	**3**
Conferencing Software for the Web	20	**20**	16
Real-Time Multimedia Web	20	19	2
The VRML Repository	20	7	16
Videoconférence et téléconférence	20	9	2
Introduction to the Internet MBONE	20	19	2
RealNetworks Home of Streaming Media	19	2	9
Sécurité - Firewalls, Cryptographie	17	**17**	9

Tableau IV: Classement avec termes d'indexation automatique seulement

Si l'on considère seulement les cas où le premier concept est correct, on a une précision de 60%. Si l'on considère les deux premiers concepts les plus approchés, la précision grimpe à 72%.

6.3.4 Classement en considérant seulement les mots-clefs

La première série d'expériences faisait intervenir exclusivement les mots-clefs dans le classement des documents, au lieu de faire appel aux termes d'indexation automatique. Par contre, ces mots-clefs n'avaient pas encore été édités par l'usager, ils représentaient le résultat de l'extraction faite par le Classeur au moment de la construction des structures internes. Les mots-clefs produits automatiquement par le prototype ne sont pas d'un grand secours pour le classement des documents. Les concepts 0 et 1, par exemple, bien qu'essentiellement vides, recueillent la majorité des suffrages pour la majorité des documents. (Voir note au dernier paragraphe de 6.1.3 pour la description de 0 et 1).

Ces expériences préliminaires démontrent l'importance pour l'usager d'ajouter ses propres mots-clefs à ceux créés automatiquement par le prototype à partir du texte du fichier de curriculum normalisé. Nous avons procédé à une édition rapide des mots-clefs pour voir si on peut effectivement les faire intervenir au moment du classement. Nous en avons profité pour éliminer les mots-clefs des concepts 0 et 1. Les succès, soit au premier soit au second concept choisi, sont indiqués par des caractères gras. Les échecs sont pourtant encore nombreux. Examinons un exemple.

L'un des sujets du curriculum, le concept 2, était "Historique de l'Internet". Nous n'avons pas réussi à cerner ce concept de façon convenable avec des mots-clefs: le système persiste à attacher la plupart des documents historiques aux concepts 5 - "Les services: courrier électronique, news, Usenet, FTP, Wais, telnet, WWW" et 8 - "HTML". Le problème est double: d'abord, il s'agit ici de juger qu'un certain document a une saveur historique plutôt que technique, c'est un problème de caractérisation de document; ensuite, l'Internet est jeune et il est difficile de choisir des termes tellement désuets qu'ils n'apparaîtraient normalement que dans un document historique.

URL	**Concept Cible**	**Concepts Choisis**
A Brief History of the Internet	2	5 et 9
The History of the Internet	2	**2 et 8**
History of the Internet and Web	2	**2 et 5**
Roads and Crossroads of Internet History	2	**2 et 5**
Internet and World Wide Web History	2	8 et 9
Internet History	2	5 et 8
Internet Society ... History of the Internet	2	8 et 5
History of the Internet and the WWW	2	8 et 5
Yahoo! Computers and Internet: History	2	5 et 8

NetHistory	2	12 et 22
... the Speed ... comparison of data rates	3	**3**
Introduction au réseau Internet	4	5 et 6
ISPs.com -- High Speed Modems	3	**3 et 5**
Organisation de l'Internet	3	12 et 21
Survol historique -- Modèle client serveur	3	7 et 4
Reducing WWW Latency ...	21	8 et 6
The Rodeo Group	21	20
Organisation matérielle du réseau	3	**5 et 3**
Conferencing Software for the Web	20	14 et 8
Real-Time Multimedia Web	20	**20 et 13**
The VRML Repository	20	**20 et 16**
Videoconférence et téléconférence	20	8 et 6
Introduction to the Internet MBONE	20	**20**
RealNetworks Home of Streaming Media	19	**7 et 19**
Sécurité - Firewalls, Cryptographie	17	**17 et 7**

Tableau V: Classement avec les termes d'indexation manuelle seulement

Si l'on considère seulement les cas où le premier concept est correct, on a une précision de 36%. Si l'on considère les deux premiers concepts les plus approchés, la précision ne grimpe qu'à 44%. Le classement ainsi opéré est nettement moins précis que celui basé sur les termes d'indexation automatique seulement. Vaut-il bien la peine de travailler avec ces termes d'indexation manuelle? Deux raisons principales m'y avaient amenée:

- ils permettent au Classeur de rattacher un document à un concept qui était jusqu'alors sans document
- ils supportent une forme simple de rétroaction.

Nous avons deux mesures de ressemblance, comment allons-nous les combiner à l'usage? Nous étudions cette question à la section 6.3.5 pour ensuite parler brièvement de l'effet de la rétroaction sur le classement, à la section 6.3.6.

6.3.5 Classement en combinant les deux jugements de ressemblance

Les deux mesures de ressemblance fournissent un indice numérique de proximité entre un concept et un document, mais leur valeur réside surtout dans l'ordre que cet indice induit sur l'ensemble des concepts. C'est donc sur les rangs correspondants que portera le raisonnement de combinaison des deux mesures. De plus, comme nous l'avons vu en 6.3.3 et 6.3.4, le classement utilisant les termes d'indexation automatique est plus précis, pour une collection naissante, que ne l'est celui basé sur les termes d'indexation manuelle. Nous combinerons ces deux rangs de la manière suivante:

- le concept le plus proche sera choisi parmi les deux premiers concepts retenus avec les termes automatiques, C1 et C2.
- si C1 ne figure pas parmi les deux premiers concepts dans l'ordre manuel mais que C2 y figure, on choisit C2.
- sinon, on choisit C1.

Les résultats obtenus en appliquant ces critères sont reproduits dans le tableau VI. Ici encore, un nombre en caractères gras indique un classement correct.

URL ou Titre	**Concept Cible**	**Premier auto**	**Second auto**	**Premiers manuels**	**Résultat**
A Brief History of the Internet	2	**2**	7	5 et 9	**2**
The History of the Internet	2	**2**	7	**2 et 8**	**2**
History of the Internet and Web	2	**2**	7	**2 et 5**	**2**
Roads and Crossroads of Internet History	2	**2**	7	**2 et 5**	**2**
Internet and World Wide Web History	2	**2**	19	8 et 9	**2**
Internet History	2	**2**	7	5 et 8	**2**
Internet Society ... History of the Internet	2	**2**	19	8 et 5	**2**
History of the Internet and the WWW	2	7	**2**	8 et 5	7
Yahoo! Computers and Internet: History	2	**2**	19	5 et 8	**2**
NetHistory	2	**2**	19	12 et 22	**2**

... the Speed ... comparison of data rates	3	**3**	2	**3**	**3**
Introduction au réseau Internet	4	9	2	5 et 6	9
ISPs.com -- High Speed Modems	3	**3**	7	**3 et 5**	**3**
Organisation de l'Internet	3	**3**	2	12 et 21	**3**
Survol historique -- client serveur	3	9	**3**	7 et 4	9
Reducing WWW Latency ...	21	**21**	19	8 et 6	**21**
The Rodeo Group	21	19	2	20	19
Organisation matérielle du réseau	3	2	**3**	**5 et 3**	**3**
Conferencing Software for the Web	20	**20**	16	14 et 8	**20**
Real-Time Multimedia Web	20	19	2	**20 et 13**	19
The VRML Repository	20	7	16	**20 et 16**	16
Videoconférence et téléconférence	20	9	2	8 et 6	9
Introduction to the Internet MBONE	20	19	2	**20**	19
RealNetworks Home of Streaming ...	19	2	9	**7 et 19**	2
Sécurité - Firewalls, Cryptographie	17	**17**	9	**17 et 7**	**17**

Tableau VI: Classement combinant les deux mesures de ressemblance

La précision est de 64% en combinant les deux mesures de ressemblance. Rappelons que le classement basé sur les seuls termes d'indexation automatique était déjà de 60%. Peut-être y aurait-il de meilleures façons de combiner les deux jugements? Notons que si nous examinons ces classements et comptons les cas où soit le classement manuel soit le classement automatique retourne le concept correct, soit au premier soit au second rang, nous obtenons une précision de 88%: il est évidemment impossible de programmer une telle décision mais nous savons au moins que les données { Terme, Poids } contiennent assez d'information pour la supporter.

Faisons maintenant jouer la rétroaction pour voir si la précision du classement augmente véritablement.

6.3.6 Classement avec rétroaction

Quand un document est ajouté par l'usager à la collection, il se produit une rétroaction *implicite*: le nouvel index verra les poids des termes d'indexation automatique modifiés par cet ajout. Mais nous présentons d'abord les résultats obtenus pour la rétroaction *explicite*, celle apportée directement sur les poids des termes d'indexation manuelle. Nous avons mentionné rapidement la rétroaction au chapitre II:

il s'agit de modifier les poids des termes pour rapprocher le document du concept jugé correct par l'usager. Pour bien isoler les facteurs en présence, nous avons fait les expériences à partir de la ligne de commande et nous avons modifié les poids sans ajouter les documents à la collection.

Rappelons que les stratégies de base pour la rétroaction consistent en l'augmentation des poids qui mèneraient à un jugement correct et la diminution de ceux qui mèneraient à un jugment incorrect. Prenons l'exemple de la colonne intitulée Rétroaction D0, dans le tableau VII. Sachant que le document D0 aurait dû être classé sous le concept 2, le système examine tous les termes d'indexation manuelle qui apparaissent à la fois dans D0 et dans C2; pour chacun de ces termes, si son nombre d'apparitions dans D0 est > 1, on augmente son poids de un dans la description de C2. Les entrées de cette colonne représentent donc les jugements de classification du système après la repondération.

URL	**Concept Cible**	**Initial**	**Rétro-action D0**	**Rétro-action D13**	**Rétro-action D18**
A Brief History of the Internet (D0)	2	5 et 9	**5 et 2**	**5 et 2**	**5 et 2**
The History of the Internet	2	**2 et 8**	**2 et 8**	**2 et 8**	**2 et 8**
History of the Internet and Web	2	6 et 5	6 et 5	6 et 5	6 et 5
Roads and Crossroads of Internet History (D3)	2	**8 et 2**	6 et 8	6 et 8	6 et 8
Internet and World Wide Web History	2	8 et 9	8 et 9	8 et 9	8 et 20
Internet History	2	5 et 8	5 et 8	5 et 8	5 et 8
Internet Society ... History of the Internet	2	6 et 8	6 et 8	6 et 8	6 et 8
History of the Internet and the WWW	2	8 et 5	8 et 6	8 et 6	8 et 6
Yahoo! Computers and Internet: History	2	6 et 5	5 et 8	6 et 5	6 et 5
NetHistory	2	12 et 22	12 et 22	12 et 22	12 et 22
... the Speed ... comparison of data rates	3	**3**	**3**	**3**	**3**
Introduction au réseau Internet	4	5 et 6	5 et 6	5 et 6	5 et 6
ISPs.com -- High Speed Modems	3	**3 et 5**	**3 et 5**	**3 et 5**	**3 et 5**
Organisation de l'Internet (D13)	3	12 et 21	12 et 21	**12 et 3**	**12 et 3**
Survol historique -- client serveur	3	8 et 7	8 et 7	8 et 7	8 et 7
Reducing WWW Latency ...	21	8 et 6	8 et 6	8 et 6	8 et 6
The Rodeo Group	21	20	20	20	20
Organisation matérielle du réseau	3	**5 et 3**	**5 et 3**	**5 et 3**	**5 et 3**
Conferencing Software for the Web (	20	14 et 8	14 et 8	14 et 8	**14 et 20**

D18)					
Real-Time Multimedia Web	20	**20 et 13**	**20 et 13**	**20 et 13**	**13 et 20**
The VRML Repository	20	**20 et 16**	**20 et 16**	**20 et 16**	**16 et 20**
Videoconférence et téléconférence	20	8 et 6	8 et 6	8 et 6	8 et 6
Introduction to the Internet MBONE	20	**20**	**20**	**20**	**20**
RealNetworks Home of Streaming ...	19	**7 et 19**	**7 et 19**	**7 et 19**	**7 et 19**
Sécurité - Firewalls, Cryptographie	17	**17 et 7**	**17 et 7**	**17 et 7**	**17 et 7**

Tableau VII: Classement avec rétroaction

L'examen du tableau VII montre qu'en fait la précision a peu augmenté: D0 est maintenant mieux classé en termes d'indexation manuelle, mais par contre D3 est plus mal classé. L'examen de D3 montre peu de recoupements entre les termes de D3 et ceux du concept Historique de l'Internet (concept 2): *WWW*, *evolution*, *applications* et *ARPANET*; de tous ces termes, seul ARPANET est vraiment représentatif du concept. La seule manière d'améliorer le classement pour D3 serait d'ajouter des termes d'indexation manuelle à la description du concept 2, termes jugés représentatifs à la fois de D3 et de C2. Le Classeur actif se limite à modifier les poids des termes.

Dans tous les cas, la rétroaction améliore le classement du document particulier pour lequel on la déclenche; mais cette rétroaction n'améliore pas le classement de documents "similaires".

6.4 Discussion sur les requêtes

La qualité des requêtes émises par le Classeur actif, bien qu'acceptable en général, est tout de même limitée de deux façons: *la restriction la plus forte à l'heure actuelle est le manque d'opérateurs booléens ou plus généralement le manque d'utilisation des fonctions de recherche avancée offertes par les divers moteurs de RI.* Il est vrai qu'à cette richesse d'expression correspond une complexification de l'interface usager mais nous pourrions munir l'agent du Classeur de cette fonctionnalité sans la faire remonter jusqu'à l'IU. Une autre restriction est *le problème d'homonymie*: si, sur soixante documents retournés pour une requête sur les noeuds, la moitié tombent dans le domaine de la météorologie marine, on perd la chance d'examiner trente documents.

On peut croire toutefois que l'approche est fondamentalement raisonnable. Elle est sujette à amélioration, mais ces améliorations sont pour la plupart une question de modifications mineures. On peut par

exemple envisager d'utiliser les mots-clefs fournis par l'usager, non seulement avec des poids positifs mais aussi avec des poids négatifs: ceci permettrait, de concert avec des requêtes booléennes, d'éviter les homonymes. Enfin, au lieu de seulement modifier les poids en cours d'apprentissage, on pourrait utiliser des expressions trouvées dans les titres des documents pertinents pour enrichir le vocabulaire de requête pour un concept donné.

Un dernier problème important relatif aux requêtes est celui que nous appellerons la saturation: au bout d'un certain nombre d'itérations, on peut imaginer que le Classeur actif ne retournera plus aucun document nouveau. Mais ce problème d'est pas tant une limitation de notre approche qu'une propriété de l'espace de recherche dans lequel se situe un usager explorant l'Internet. Si vraiment l'usager désire trouver d'autres documents, on pourrait lui fournir divers opérateurs de modification des requêtes, évoqués au dernier chapitre VII. La latéralisation des requêtes, en particulier, semble prometteuse à cet égard.

6.5 Discussion sur le classement

Tel qu'évalué à 60% ci-dessus, la précision du classement apporté par le Classeur actif semble satisfaisante si l'on tient compte qu'il s'agit du classement d'un document parmi les multiples catégories représentées par l'arbre des concepts, plutôt qu'un simple Pertinent/Non-Pertinent, ou encore un Chaud/Tiède/Froid comme dans Syskill & Webert. Toutefois, il est clair que le classement basé sur les termes d'indexation manuelle est très sensible à la qualité de ces termes d'indexation en termes de séparation entre les concepts et de couverture de chaque concept. La rétroaction explicite est un peu décevante mais la rétroaction explicite, par ajout de documents à l'index, compense ces lacunes.

6.5.1 Concepts sans documents

Les concepts sans documents sont certainement plus difficiles à enrichir que les autres concepts: le processus de formulation des requêtes est identique mais, au moment du classement, les règles de combinaison des résultats telles qu'énoncées en 6.3.5 vont défavoriser de tels concepts. Par contre, l'usager peut choisir parmi trois modes de classement: index automatique seulement, index manuel seulement et combinaison des deux index. Dans le contexte qui nous occupe, il devrait choisir le classement par index manuel. De plus, le Classeur devrait assouplir ses règles de combinaison pour distinguer entre un score de 0.0 en l'absence de documents et un autre en l'absence de recoupements.

6.5.2 Documents sans concepts

Pour l'instant, le Classeur actif ne fait pas de seuillage pour la pertinence des documents: c'est une lacune qu'il faudrait combler. Quiconque se sert d'un moteur de RI pour trouver des documents sur des sujets en apparence anodins recevra un jour ou l'autre les URL de sites pornographiques, l'Internet étant très pollué à l'heure actuelle. La mise en place d'un seuil minimal de pertinence, combinée à la possibilité pour l'usager d'exclure des sites de ses recherches, pourrait pallier au problème de pollution. Quant au problème d'homonymie, par exemple BD pour Base de Données et Bandes Dessinées, la meilleure façon de le régler serait l'enrichissement de l'index manuel, qui n'est pas une tâche facile.

6.6 Cycle de vie d'une collection gérée par le Classeur actif

Au terme de ces expériences avec le Classeur, j'ai une meilleure perception de la réalité de son usage quotidien et de l'évolution d'une collection. *Je décrirai ici brièvement la naissance, la croissance et la maturité d'une collection gérée par le Classeur (dans son état actuel de développement).*

Au commencement il y a un univers conceptuel encore peu structuré. L'usager externalise cet univers sous la forme d'un arbre de concepts ou de sujets et acquiert quelques documents et quelques mots-clefs se rattachant à certaines des branches ou des feuilles de cet arbre. Il possède maintenant assez de données pour construire la structure de base passée en entrée au Classeur actif, cette structure que nous avons appelée un curriculum. Il invoque maintenant le Classeur pour la première fois et le Classeur construit et sauvegarde les structures internes dont il a besoin, notamment l'index automatique et la structure de noeuds et de liens.

L'usager peut maintenant utiliser son Classeur actif pour enrichir la collection naissante. Il fait des demandes d'enrichissement pour un concept donné, soit interactivement soit en mode lot, pour exécution ultérieure. S'il travaille interactivement, il attend un moment et reçoit une liste de documents résultant d'une requête. Il sélectionne un document dont l'URL paraît intéressant et demande son classement. Puis il l'examine pour accepter ou modifier le classement. Le document est intégré à la collection et accessible à-travers le Classeur. S'il travaille en mode lot, il peut examiner subséquemment le rapport produit par le

Classeur: ce rapport présente la liste des documents repérés ainsi que le classement obtenu pour chacun. Le rapport permet d'examiner les documents et de juger ainsi de la justesse du classement. L'usager peut alors modifier directement les structures internes du Classeur; ou bien il peut demander au Classeur de regénérer un curriculum HTML externe sur la base de l'état courant des données. Ou enfin il peut ouvrir le Classeur graphique et ajouter le document interactivement à sa collection.

Durant cette étape de la vie de la collection, le Classeur découvre de nouvelles ressources pour l'usager, l'usager les intègre ou les rejette. L'usager améliore ses mots-clefs pour chaque concept. L'usager modifie sa structure de concepts à travers la fission ou la fusion de concepts (voir chapitre VII). Le Classeur améliore son classement à travers la rétroaction explicite et surtout implicite. Il améliore son filtrage au moyen de la liste de sites rejetés par l'usager. Le Classeur peut actuellement supporter jusqu'à 60 concepts dans une seule collection, mais l'affichage deviendra moins utile en l'absence de mécanismes hiérarchiques ou autres.

Finalement, le Classeur deviendra de moins en moins capable de découvrir automatiquement des nouvelles ressources pour l'usager: les moteurs de RI de la configuration courante auront déjà retourné tous les documents pertinents des domaines IP couverts par leurs robots (peu importe la qualité des requêtes). L'usager peut alors configurer un nouveau moteur de RI. Ou encore il peut continuer d'utiliser la collection pour ses mécanismes de classement en allant lui-même chercher d'autres documents au hasard de ses périples sur le Web. Ou enfin, son vocabulaire étant enrichi par l'expérience, il va créer une nouvelle collection, une nouvelle carte conceptuelle, pour mieux explorer l'une des régions déjà abordées.

6.7 Conclusion

L'évaluation finale de validité du Classeur devrait se faire par comparaison avec les requêtes et le classement potentiellement fournis par l'usager. Notre objectif n'était pas de construire un système qui puisse remplace l'usager - lire "intelligent" - mais plutôt un assistant pour lui faciliter la tâche. Cette évaluation devrait aussi tenir compte de la qualité des SRI publics sur Internet.

Des études menées dans le cadre d'un atelier spécial TREC sur le Web, en 1999, semblent démontrer que la qualité de ces SRI est inférieure à celle des systèmes traditionnels [Hawk99]. Pour cet atelier, on a

préparé un corpus d'évaluation, le VLC2, devant permettre de comparer la performance des SRI traditionnels participant à TREC avec celle des moteurs de RI publics disponibles sur l'Internet. Le corpus provient d'une archive Internet comprenant plus de 18 millions de pages. Les résultats de la comparaison montrent une précision P@20 (pour les 20 premiers documents repérés) allant de 0.23 à 0.38 pour cinq SRI sur Internet alors que leur rivaux TREC offrent une médiane allant de 0.40 à 0.61, selon la quantité de données dans chaque document utilisée pour répondre aux requêtes.

Notons enfin que, puisque l'IU permet l'addition à une requête de termes qui n'affecteront que la présente requête, la fonctionnalité de requête est la même que celle disponible à-travers un navigateur standard, dans la mesure où la majorité des usagers ne font jamais appel aux fonctions de recherche avancées. *Certains travaux récents cités dans le même article font état d'une moyenne de 2.35 termes par requête pour les SRI publics sur Internet, avec des opérateurs booléens utilisés dans moins de 10% des requêtes.*

Chapitre VII: Recherches futures

Un projet comme le Classeur est circonscrit dans le temps: nous avons seulement implanté un petit noyau de fonctionnalité, tel que décrit par les classes Java aux chapitres V et VI. Quelles sont les principales lacunes du Classeur actif? Nous souhaiterions donner au Classeur plus de souplesse et plus de suivi dans la formulation des requêtes. Nous aimerions offrir un peu plus de services en ce qui regarde la structure de concepts. L'univers des documents devrait être élargi et approfondi à la fois. Finalement, l'interface graphique est encore primitive, bien qu'adéquate. Nous aimerions étendre ou transformer le Classeur le long de ses trois axes: requêtes, documents et concepts (Figure 7-1). Dans ce dernier chapitre, nous décrirons ces améliorations, les unes faciles, les autres ambitieuses. Nous terminerons en situant le Classeur dans l'univers présent de la gestion de l'information et en extrapolant vers l'univers futur mais proche de la gestion des connaissances et des systèmes d'information intelligents.

7.1 Amélioration des requêtes

Le Classeur ne réfléchit guère sur les résultats de ses requêtes. Si l'une des requêtes échoue à repérer des documents pertinents, le Classeur graphique ne s'en rend pas compte; de même si l'un des SRI est continuellement hors-ligne ou si l'autre fournit plus de réponses intéressantes. La raison est simple: le Classeur graphique n'a pas de mémoire. Cette lacune est due au fait que le Classeur graphique utilise la classe *RequeteClasseur* pour ses opérations: cette classe, très limitée, prend en entrée une chaîne de caractères et le nom d'un SRI; elle ne sait comment reformuler une requête ni sous quelles conditions il serait avantageux de le faire.

7.1.1 Gestion des requêtes

Il manque donc au Classeur graphique une couche de gestion des requêtes: nous désignons ainsi tout le traitement qui consiste à savoir que l'on a déjà émis une requête pour un concept donné, que l'on avait formulé la requête de telle façon et qu'elle avait donné tel résultat. C'est dans cet objectif de contrôle et de gestion que nous avons conçu la classe *RequeteConcepts*. Il ne lui manque que la comparaison de ses "nouveaux" documents avec ceux déjà présents ou encore ceux déjà rejetés. Il lui manque aussi des façons

de modifier ses requêtes quand elles ne donnent pas des résultats satisfaisants. Enfin, le Classeur ne l'utilise couramment qu'en mode lot, non graphiquement.

7.1.2 Opérations sur les requêtes

Après avoir lancé une requête et examiné les résultats, l'usager ou la classe *RequêteConcepts* aimerait peut-être avoir une façon simple de modifier quelque peu la requête, transformant ainsi la liste des résultats. Nous envisageons trois opérations sur les requêtes, la généralisation, la spécialisation et la latéralisation.

- La *généralisation* est définie comme une façon de relancer le filet de la requête pour attraper plus de documents. La façon la plus simple de généraliser une requête serait de lui enlever des mots. Une autre façon serait de la reformuler logiquement avec des OU plutôt que la syntaxe d'expression que nous utilisons par défaut. Une troisième façon serait de généraliser les termes de la requête eux-mêmes, en utilisant pour cela un thesaurus ou une ontologie. Finalement, on pourrait suivre les liens au moment du filtrage.

- La *spécialisation* est une façon de resserrer le filet. Si l'on veut augmenter la précision, on peut ajouter des mots, reformuler avec des ET, ou bien spécialiser les termes eux-mêmes. Il est clair que les requêtes à un seul mot sont à éviter: elles sont plus sujettes aux problèmes d'homonymie. On peut exploiter les termes d'indexation manuelle ou les documents déjà attachés au noeud pour y glaner des mots, dans les titres par exemple. On peut aussi, pour certains engins, spécifier des sites intéressants. Enfin on peut, au moment du filtrage des documents reçus, éliminer certains sites: nos classes de requête offrent déjà cette possibilité indispensable.

- Enfin, on peut souhaiter repérer des documents pertinents pour un concept mais en utilisant une forme de pensée latérale ou d'association d'idées pour définir ce concept. Nous désignons par *latéralisation* un déplacement dans un arbre qui n'est ni strictement vers le haut, ni strictement vers le bas. L'une des façons de latéraliser une requête serait de remplacer ses termes par des termes équivalents dans une autre langue. Une autre façon de latéraliser serait de monter d'un niveau puis de redescendre d'un niveau. L'utilisation de thesauri - avec cette fois leurs liens latéraux - et le traitement translinguistique permettraient de supporter une reformulation des requêtes.

7.1.3 Enrichissement du langage de requêtes

Nous avons parlé de gestion des requêtes et d'opérations sur les requêtes. Mais il faut souligner ici que la plus grande lacune du Classeur de ce côté est de ne pas donner accès à toute la richesse du langage de requêtes, à toutes les options des SRI utilisés. Pour éviter la complexification de l'IU [Shne98] nous pensons enrichir le langage de requêtes strictement à l'interne et utiliser cette fonctionnalité dans les opérations sur les requêtes. Pour un usager spécialiste en requêtes, le Classeur demeure utile car ses formats de fichiers d'entrée et de sortie sont délibérément ASCII. Un usager spécialiste pourrait donc formuler des requêtes très complexes, sauvegarder les pages de réponses et en nourrir le Classeur.

7.2 Traitement des concepts

Il y a donc fort à faire du côté du traitement des requêtes, pourtant, le traitement de base est déjà là. Il n'en est pas de même pour une autre des facettes de cet univers: le Classeur actif néglige les concepts. Nous envisageons la création d'un environnement, un hyper-éditeur - qui prenne en compte connaissances et ressources comme deux facettes d'un même univers; ceci faciliterait la création de liens entre ces deux modèles ainsi que le va-et-vient des perspectives qui constitue une part importante du travail. Pour ce faire, il faudrait d'abord enrichir le traitement des concepts. Nous aimerions supporter la définition, la fission et la fusion des concepts.

7.2.1 Définition d'un concept

Le Classeur actif n'est pas un environnement de création de tutoriels. Toutefois, il est capable d'importer un curriculum, il peut aussi exporter et il permet certaines modifications aux noeuds-concepts. Par exemple, on peut avec le Classeur modifier la liste des documents attachés à un concept; on peut aussi modifier les termes d'indexation manuelle. Ces termes d'indexation sont un type d'annotation du concept, qui évoquent des entrées dans un thesaurus. Or, pour *chaque* champ d'un concept, on devrait pouvoir supporter son édition et on devrait pouvoir imposer les contraintes et vérifications sémantiques globales entraînées par sa modification. Au lieu de supporter seulement Nom, Terme et Doc, on pourrait supporter bien d'autres champs. L'un des champs les plus intéressants à supporter serait un genre de Lien, qui contiendrait non pas des termes mais des références à d'autres concepts, dans d'autres cours ou d'autres

univers conceptuels. On peut aussi imaginer des champs à saveur plus pédagogique, comme Prérequis, Stratégie, Notes de cours, ainsi de suite.

7.2.2 Fission

Le Classeur pourrait supporter la fission d'un concept en sous-concepts. *Au lieu de se borner à classer des documents sous des concepts existants, il pourrait utiliser les indices apportés par le classement des documents pour suggérer d'extraire un sous-concept.* (Notons que les documents ne sont pas attachés qu'aux feuilles de l'arbre, ils peuvent être attachés partout). Comment et pourquoi déciderait-on de créer un sous-concept? Cela pourrait se produire au terme d'une analyse des grappes de documents, par exemple. Supposons que le Classeur, au terme d'une analyse par grappes, découvre deux sous-groupes de documents bien distincts rattachés au concept X. Il y aurait lieu d'examiner ces deux sous-groupes pour voir si on ne devrait pas en faire des sous-concepts un peu plus permanents. La découverte des sous-groupes peut se faire de plusieurs façons, par exemple [Rasm92]. Par contre, *l'étape de loin plus difficile est de tenter de nommer ces sous-groupes* pour offrir à l'usager des étiquettes raisonnables comme point de départ de son analyse.

7.2.3 Fusion

Le Classeur pourrait aussi supporter la fusion de concepts. Précisons par un exemple. Supposons que les concepts frères A et B se ressemblent beaucoup et que le concept C, aussi attaché au même parent X, est tout à fait à part. (On découvre des ressemblances et différences par un examen de l'index, graphique au besoin, ou encore au terme d'une analyse par grappes.) Le Classeur pourrait alors suggérer à l'usager l'existence d'un concept Y, entre X et A, B: C deviendrait ainsi l'oncle, plutôt que le frère, de A et B. *En bref, la perception des ressemblances et la perception des distinctions, qui sont probablement les deux opérations mentales les plus importantes, peut à coup sûr être supportée par ces approches.*

7.3 Univers des documents

Nous avons évoqué les lacunes du Classeur dans le domaine du traitement des requêtes et dans celui du traitement des concepts. Il est temps de passer à la troisième et dernière facette de cet univers, la facette

des documents. Nous discuterons des améliorations dans l'ensemble des documents-cibles et nous évoquerons les possibilités de traitement des documents individuels.

7.3.1 Formats autres que HTML

Bien que le format HTML représente une proportion substantielle des documents accessibles sur le Web, la myopie du Classeur face aux autres formats de fichier est un handicap de taille. Le format PostScript, par exemple, échappe à nos procédures d'indexation actuelles. On pourrait envisager d'utiliser ou de construire des outils pour indexer des fichiers PostScript, pour en extraire des termes d'indexation déjà présents ou encore pour exporter un format PostScript dans un format texte. Notons qu'on peut quand même attacher les documents PostScript à des concepts du curriculum; le seul problème est qu'ils n'ont aucun impact sur l'index automatique.

7.3.2 Granularité et couverture du repérage

Le Classeur ne récupère actuellement que les URL au niveau zéro, ceux directement présentés par le SRI. Il y aurait lieu de mettre en place une certaine logique utilisant la caractérisation de ces URL: lorsqu'ils ne constituent qu'un genre de table des matières d'un document composé, il faudrait récupérer d'autres URL pour l'analyse. Il faudrait donc, dans ce cas, repérer et analyser plus de texte. Le repérage et l'analyse des URL au niveau un - c'est-à-dire l'étude les liens dans les documents repérés par le SRI externe - serait aussi intéressante et importante parce qu'elle aiderait à pallier le manque de couverture des SRI externes. Le Web croît actuellement beaucoup plus vite que ce que les spiders peuvent trouver et ils utilisent un jugement d'utilité/popularité qui n'est pas toujours le nôtre. Par contre, il serait aussi intéressant de repérer des passages de documents; certains travaux dans la littérature des SRI portent déjà sur cette possibilité, par exemple [Salt90]. On peut découper un document en passages au moyen d'une analyse de l'usage des termes: il est alors possible de présenter à l'usager les passages les plus pertinents.

Pour l'instant, le Classeur ne fait que classer et permettre l'accès aux documents repérés. Nous suggérons ci-dessous quelques améliorations au traitement: la plupart font intervenir des traitements linguistiques ou statistiques plus profonds. Nous envisageons de caractériser les documents, de les résumer et de tenter d'analyser leur structure.

7.3.3 Caractérisation et construction de sommaire

L'objectif de l'opération de caractérisation est de calculer un certain nombre de caractéristiques du document et de les faire connaître à l'usager au moyen d'une présentation visuelle. On trouve ce genre de traitement dans certains travaux de Pirolli [Piro96]. La littérature récente en sciences de l'information ou en bibliothéconomie offre aussi des pistes (les facettes, MDS). Finalement, beaucoup de travaux ont porté sur la sommarisation des documents: nous croyons que les approches par extraction de phrases, bien que leur résultat manque d'un certain poli linguistique, sont relativement faciles et représenteraient un apport important [Smit90]. Il faudrait toutefois que l'usager soit conscient des limites du système.

7.3.4 Découverte de structure

On devrait offrir à l'usager, quand on peut l'extraire des balises HTML ou autres, un arbre représentant la structure du document. En termes d'implantation, nos classes sont engagées dans cette voie; il suffirait de relaxer les contraintes imposées au curriculum. Un autre outil intéressant serait la segmentation par formation de grappes. Il s'agit ici d'induire une structure au terme d'un traitement numérique. Certains travaux effectués dans une perspective de repérage de passages de documents [Salt90] utilisent des méthodes de grappes pour révéler, graphiquement, la structure implicite d'un document.

7.4 Interface graphique

L'interface usager présente des lacunes tant au niveau des entrées que des sorties. L'interface d'entrée de données laisse beaucoup à désirer car nous avons mis l'accent sur les traitements internes, nous concentrant, avec l'IU, sur les sorties plutôt que les possibilités d'entrées. Parlons maintenant des sorties du Classeur.

Quelles sont les opérations visuelles à supporter dans le *panneau-réseau*? Nous croyons qu'il faut supporter des interactions comme: cliquer sur un noeud pour avoir des détails, rendre des noeuds invisibles, centrer sur un noeud, montrer seulement les noeuds qui sont à distance <= N d'un certain noeud et configurer les attributs visuels (taille, forme, couleur). La principale limitation est la taille de l'espace visualisable et le fait qu'on ne puisse couramment passer d'une collection à une autre, d'un univers conceptuel à un autre. Bien que trente à quarante concepts suffisent pour baliser l'espace d'un seul cours, ils sont nettement insuffisants pour supporter l'exploration des intérêts d'un usager.

Les sorties HTML du Classeur actif sont encore rudimentaires: nous les avons surtout créées pour refléter les résultats du traitement par lot. Ces sorties HTML sont un ajout relativement récent, au départ les sorties étaient faites simplement dans un fichier texte pouvant être directement réutilisé comme un fichier DAT, que l'on pense aux documents repérés, par exemple. La production de fichiers HTML offre une façon plus élégante de relier les traitements mode-lot et les traitements interactifs. Finalement, le fichier d'aide du Classeur actif, rédigé manuellement, est encore primitif; nous aimerions lui ajouter des rubriques et du graphisme.

Toutes les améliorations mentionnées dans les sections 7.1 à 7.4 peuvent s'intégrer facilement dans l'architecture actuelle du Classeur actif, elles ne constituent que des extensions. Elles décrivent un espace de transformations allant de la formulation en Java d'algorithmes simples - par exemple, repérer les URL au niveau un - jusqu'à l'analyse et la conception de fonctionnalité complexe, ou l'intégration de bibliothèques existantes, pour la découverte de structure. Nous terminerons la description de notre problème de gestion d'information en prenant assez de distance pour tenter d'envisager la gestion des connaissances.

7.5 Vers la gestion des connaissances

J'ai présenté dans ce mémoire un problème de gestion d'information et une solution avec un prototype Java. Pour conclure, il faut essayer de replacer le Classeur actif dans le contexte général de la quête d'information. Quand j'ai parlé, au chapitre II, des processus de quête d'information, j'ai repris les catégories et dimensions de Oard et Marchinioni [Oard96]. Au terme de ce mémoire, je discuterai des facettes qui me semblent les plus importantes dans le contexte du support à la tâche d'un usager. Ces facettes, représentées à la figure 7-2, sont la cueillette, le filtrage, les requêtes, l'extraction d'information, la structuration, la visualisation, la navigation et enfin les systèmes d'information intelligents.

- Cueillette et filtrage: Le filtrage d'information se distingue de la RI par l'accent mis sur la sélection, à partir d'un flot dynamique de documents, de textes susceptibles de combler un besoin d'information relativement stable et bien défini. Le filtrage peut être individuel ou communautaire. Le Classeur est sans contredit un outil de filtrage d'information. Les divers espaces de concepts y jouent le rôle de profils d'usager. Quant à la cueillette d'information, il faudra l'envisager dans un sens plus large englobant désormais des modèles push (initiative-serveur) autant que des modèles pull.

- Requêtes: Au début de l'histoire des SRI, la formulation d'une requête nécessitait la connaissance de la hiérarchie des sujets dans un domaine donné; on peut maintenant formuler des requêtes textuelles, avec certains opérateurs booléens et des opérateurs de proximité. Dans le futur, on pourra de plus en plus demander et retrouver des ressources dans un langage conceptuel; le calcul de correspondance prendra lui aussi un aspect plus sémantique, en utilisant divers outils comme des thesauri et des ontologies permettant de construire des passerelles entre divers univers linguistiques. Le Classeur actif se situe tout au début de cette route, il est orienté-concept.

- Extraction et prospection: L'extraction d'information s'appuie sur un TLN plus profond. Il s'agit de saisir non plus le thème, mais bien ce que dit le document à propos de son thème. Normalement, on construira manuellement une palette de patrons d'activation. Chaque patron d'activation correspondra à un cadre conceptuel avec des rôles bien définis: il s'agira alors, en faisant jouer des connaissances lexicales, de remplir les cases du cadre conceptuel en trouvant les mots du texte qui désignent les acteurs ou leurs propriétés. Sur un autre front, Marti Hearst [Hear97] fait le point sur la promesse des méthodes statistiques et de la linguistique empirique. Elle définit les tâches principales de la prospection de données textuelles, découverte de pépites d'information, découverte de tendances et finalement visualisation des données. *Hearst reprend un argument déjà trouvé souvent dans la littérature, celui de la facilité du traitement textuel due à la grande redondance de ce type d'information: n'importe quel algorithme raisonnable est en mesure de dégrossir cette information.*

- Structuration: Par structuration de l'information, nous entendons la construction ou la mise en évidence de structures dans de grandes quantités de documents, pour faciliter l'accès ou la compréhension; elle peut être imposée de l'extérieur, avec des catégories toutes faites; elle peut être construite par une communauté d'usagers; ou enfin elle peut être induite par diverses méthodes d'apprentissage ou de prospection. Le Classeur représente une forme simple de structuration de l'information: une structure de documents et de concepts construite par un usager sert de point de départ et évolue en s'enrichissant de nouveaux documents et éventuellement de concepts et de liens.

- Visualisation: *Nul doute que la modalité visuelle ait un rôle important à jouer quand il s'agit de déceler une structure dans une masse de données, ou encore de construire cette structure.* Nous croyons que la visualisation de l'information conduira à des modes de pensée autres, tout en continuant de supporter plusieurs des modes de raisonnement courants. Les cartes conceptuelles et outils de mise en page automatique forment déjà une base intéressante. Cette évolution requiert la construction de meilleurs modèles des opérations mentales ainsi qu'une transformation radicale des interfaces. La génération des jeux vidéo ne peut que trouver insipides les interfaces actuelles.

- Navigation: On peut distinguer essentiellement deux type de navigation; il y a une navigation logique, textuelle, par liens, comme dans Netscape par exemple, et encore la navigation dans un espace virtuel de documents muni, plus ou moins artificiellement, d'une topologie et d'une géographie. La navigation dans des espaces virtuels devient de plus en plus répandue, avec l'arrivée d'ordinateurs plus puissants et de meilleurs outils pour construire ces espaces. Mais cette navigation n'est pas sans problème, le plus important étant sans doute la question de désorientation, qui est présente aussi somme toute avec la navigation logique.

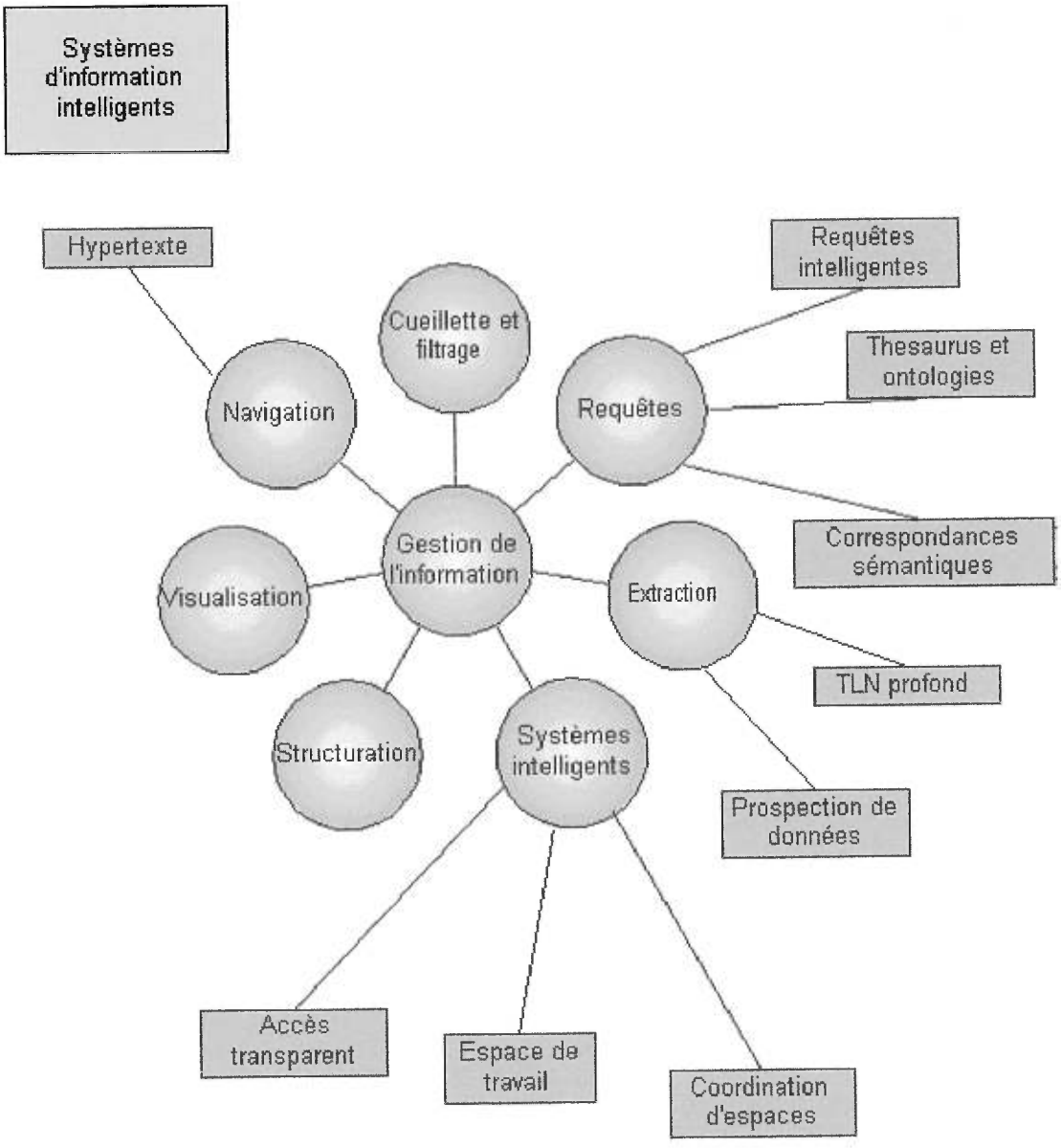

Figure 7-1: Systèmes d'information intelligents

Les traitements possibles gagnent donc chaque jour en largeur et en profondeur. Ces traitements rendent graduellement possible un nouveau type de système d'information, les systèmes d'information intelligents. Dans ces systèmes, nous croyons la navigation appelée à disparaître, sauf pour des usages très spécialisés, en faveur d'un accès transparent à l'information.

7.6 Conclusion

L'ordinateur est maintenant perçu d'abord et avant tout comme un outil d'accès à l'information; pourtant les usagers sont souvent forcés de traiter avec des outils conçus pour des spécialistes [...] *les systèmes d'information intelligents commencent à voir le jour: ce sont des systèmes orientés-tâche et la connaissance de la tâche de l'usager leur permet de s'intégrer à son environnement de façon transparente et de fournir l'information [...] au moment où elle devient pertinente, sans que l'usager ait à invoquer une fonction spécifique. [AAAI98]* La visualisation est l'un des outils qui rendent ces systèmes possibles: elle peut maintenant s'attaquer à ce que Card appelle les outils de compréhension et à l'espace du travail [Card96].

Hearst définit l'objectif des systèmes d'accès à l'information comme étant d'aider les gens à découvrir, créer, utiliser, réutiliser et comprendre l'information [Hear97]. Le Classeur actif s'inscrit dans cette problématique. Bien qu'il apporte peu d'éléments originaux, il représente une intégration intéressante et utile de technologies éprouvées pour aider l'usager dans sa tâche de structuration des connaissances, en plus de fournir un bon tremplin pour une intégration naturelle et puissante de technologies plus profondes. Le manque de standards dans le domaine des STI, d'abord perçu comme un obstacle, nous a finalement conduits à un système ouvert.

Sources documentaires

- [AAAI98] American Association for Artificial Intelligence, Symposium on Intelligent Environments, Spring 1998, Technical Report SS-98-02.

- [Avis97] B. Avishai, Get What You Want from the Web, Fortune, 26 octobre 1997, pp. 283-284.

- [Ball99] S. Ball, Supercharged Strings, Java Report, Février 1999, Volume 4, Numéro 2, pp. 62-69.

- [Bowm94] C.M. Bowman, P.B. Danzing, U. Manber and M.F. Schwartz, Scalable Internet Resource Discovery: Research Problems and Approaches, CACM, August 1994, Volume 37 No. 8, pp. 98-114

- [Burn98] Kathleen Burnett, E.Graham McKinley, A Rhizomorphic Model of Information Contexts, in Concluding the Special Issue on HCI and Information Retrieval, Interacting with Computers, Volume 10, Number 3, June 1998.

- [Card96] S.K. Card, Visualizing Retrieved Information: A Survey, IEEE Computer Graphics and Applications, Vol. 16, No. 2 (Mars 1996), pages 63-67.

- [Chen98] C. Chen, Generalized Similarity Analysis and Pathfinder Network Scaling, Interacting with Computers, Volume 10, Number 2, May 1998, pp.107-128.

- [Eklu95] J. Eklund, Cognitive models for structuring hypermedia and implications for learning from the world-wide web"; Proc. Ausweb95: The First Australian World-Wide Web Conference, Southern Cross University Press, Ballina, Australia (1995) 111-116. http://elmo.scu.edu.au/sponsored/ausweb/ausweb95/papers/hypertext/eklund/index.htm.

- [Falo95] C. Faloutsos et D.W. Oard, A Survey of Information Retrieval and Filtering Methods, University of Maryland CS-TR-3514, 1995, 23 p.

- [Fors86] R. Forsyth et R. Rada, Machine Learning: Expert Systems and Information Retrieval, Ellis Horwood, London, 1986.

- [Fox92] C. Fox, Lexical Analysis and Stoplists, dans Information Retrieval - Data Structures and Algorithms, W.B. Frakes et R. Baeza-Yates, Editors, Prentice Hall, 1992, 504 p.

- [Frak92] W.B.Frakes, Stemming Algorithms, dans Information Retrieval - Data Structures and Algorithms, W.B. Frakes et R. Baeza-Yates, Editors, Prentice Hall, 1992, 504 p.

- [Fric99] A. Frick, G. Sander et K. Wang, Simulating Graphs as Physical Systems, Dr. Dobb's Journal, Août 1999, pp. 58-64.

- [Gain95] B. Gaines and M. Shaw, Concept Maps as HyperMedia Components. http://ksi.cpsc.ucalgary.ca/articles/ConceptMaps

- [Guha90], R.V. Guha et D.B. Lenat, Cyc: A Mid-Term Report, AI Magazine, Vol. 11, No. 3 (Fall 1990), pp. 32-59.

- [Harm92], D. Harman, Relevance Feedback and Other Query Modification Techniques, Information Retrieval - Data Structures and Algorithms, W.B. Frakes et R. Baeza-Yates, Editors, Prentice Hall, 1992, 504 p.

- [Hawk99] D. Hawking, N. Craswell, P. Thistlewaite et D. Harman, Results and challenges in Web search evaluation, Proceedings of the Eight International World Wide Web Conference, North Holland, Mai 1999, pp. 1321-1330.

- [Hear97] M.A. Hearst, Text Data Mining - Issues, Techniques and the Relationship to Information Access, UW/MS Workshop on Data Mining, July 1997. http://www.sims.berkeley.edu/~hearst/talks/dm-talk

- [Helm96] D.J. Helm, R.J.D'Amore, P.-F. Yan, MITRE Information Discovery System, Proceedings of WebNet96. http://aace.virginia.edu/aace/conf/WebNet/html/201.htm

- [Hend95] R.J Hendley, N.S. Drew, A.M. Wood and R. Beale, 1995. Narcissus: visualising information. Proc. of the First IEEE Information Visualization Symposium (InfoVis'95). ftp://ftp.cs.bham.ac.uk/pub/authors/ R.J.Hendley/ ieeeviz.ps.Z

- [Huan98] M.L. Huang, P. Eades and J. Wang, On-Line, Animated Vizualisation of Huge Graphs Using a Modified Spring Algorithm, Journal of Visual Languages and Computing (1998) 9, 623-645.

- [Lass98] O. Lassila, Web Metadata: A Matter of Semantics, IEEE Internet Computing, Volume 2, Number 4, July-August 1998, pp. 30-37.

- [Lesk95] M. Lesk, The Seven Ages of Information Retrieval, Conférence présentée au Symposium Vannevar Bush, 12-13 octobre 1995, MIT, Boston, Massachusetts. http://www-evat.mit.edu/bush/talk6

- [Lewi96] D.D. Lewis and K. S. Jones, Natural Language Processing for Information Retrieval, Communications of the ACM January 1996, pp. 92-101.

- [Li98] Yanhong Li, Toward a Qualitative Search Engine, IEEE Internet Computing, Volume 2, Number 4, July-August 1998.

- [Lieb95] H. Lieberman, Letizia: An Agent That Assists Web Browsing, Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, Quebec, Canada, 1995, pp. 924-929.

- [Merr92] M.D. Merrill, Z. Li and M.K. Jones, Instructional Transaction Shells: Responsibilities, Methods and Parameters, Education Technology, Février 1992, pp. 5-26.

- [Mitc97] T. Mitchell, Does Machine Learning Really Work? AI Magazine, Volume 18, No 3 (Automne 1997), pp. 11-20.

- [Mizz98] S. Mizzaro, How Many Relevances in Information Retrieval? Interacting with Computers 10 (1998) pp. 303-320.

- [Mull97] P.-A. Muller, Instant UML, Wrox Press, 1997, 343 pages.

- [Nkam96] R. Nkambou et G. Gauthier, Un modèle de représentation du curriculum dans un STI, Intelligent Tutoring Systems: Third International Conference, ITS'96, Springer, Lecture Notes in Computer Science, pp. 420-429.

- [Oard96] D.W. Oard and G. Marchinioni, A Conceptual Framework for Text Filtering, University of Maryland CS-TR-3643, 1996, 32 p.

- [Paqu96] G. Paquette et J. Girard, AGD: A Course Engineering Support System, Intelligent Tutoring Systems: Third International Conference, ITS'96, Springer, Lecture Notes in Computer Science, pp. 382-391.

- [Pazi97] M.T. Pazienza (Ed.), Information Extraction, A Multidisciplinary Approach to an Emerging Information Technology, "Lecture Notes in Artificial Intelligence", Springer, 1997, 213 p.

- [Pazz96] M.J. Pazzani, J. Muramatsu et J. Billsus, Syskill and Webert: Identifying Interesting Web Sites, Proceedings of the Thirtheenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference, AAAI Press/MIT Press, Menlo Park, 1996, pp. 54-61.
  http://www.ics.uci.edu/~pazzani/RTF/AAAI.html

- [Perk97] Perkowitz, M., Doorenbos, R., Etzioni, O., Learning to Understand Information on the Internet: An Example-Based Approach, Machine Learning, 1997.
  ftp://ftp.cs.washington.edu/pub/map/papers/ILA-JIIS.ps

- [Piro96] P. Pirolli, J. Pitkow et R. Rao, Silk from a Sow's Ear: Extracting Usable Structures from the Web, Georgia Institute of Technology - Graphics, Visualization and Usability Center, Technical Report, No. 96--10.

- [Raja98] A. Rajamaran et P. Norvig, Virtual Database Technology: Transforming the Internet into a Database, IEEE Internet Computing, Volume 2, Number 2 (July-August 1998), pp. 55-58.

- [Rasm92] Rasmussen, Clustering, Information Retrieval - Data Structures and Algorithms, W.B. Frakes et R. Baeza-Yates, Editors, Prentice Hall, 1992, 504 p.

- [Rijs79] C.J. van Rijsbergen, Information Retrieval, London, Butterworths, 1975.
  http://www.dcs.gla.ac.uk/Keith/

- [Salt83] G. Salton et M. McGill, An Introduction to Modern Information Retrieval, New York, McGraw-Hill, 1983.

- [Salt90] G. Salton, J. Allan and C. Buckley, Approaches to Passage Retrieval in Full Text Information Systems, Proceedings SIGIR-93, Association for Computing Machinery, New York, June 1993, 49-58.

- [Scha96] , B. Schatz, W.H. Mischo, T.W. Cole, J.B.Hardin, A.P.Bishop et H. Chen, Federating Diverse Collections of Scientific Literature, IEEE Computer, May 1996, pp. 28-36.

- [Smit90] P.D. Smith, An Introduction to Text Processing, MIT Press, Cambridge, Massachusetts, 1990, 300 p.

- [Shne98] B. Shneiderman, D. Byrd et W.B. Croft, Sorting Out Searching - A User Interface Framework for Text Searches, Communications of the ACM, Vol 41, No. 4 (April 1998), pp. 95-98.

- [Srin92] P. Srinivasdan, Thesaurus Construction, Information Retrieval - Data Structures and Algorithms, W.B. Frakes et R. Baeza-Yates, Eds., Englewood Cliffs, Prentice Hall, 1992, 504 p.

- [ThoJ96] J. Thomson, J.E.Cooke and J.E.Greer, The MicroWeb Toolkit: Bringing the WWW to the Classroom, WebNet96 - World Conference of the Web Society, San Francisco, October, 1996. http://aace.virginia.edu/aace/conf/webnet/html/371.htm

- [Thom95] P. Thompson, West Publishing, Information Retrieval, Text Categorization and Intelligent Information Agents: Statement of Research Interest. www.cs.umbc.edu/thompson.html.

- [Thom98] B. Thomas, Rank and File, IEEE Internet Computing, Volume 2, Number 4 (July-August 1998).

- [YanG95] T.W. Yan, H. Garcia-Molina, SIFT - A Tool for Wide-Area Information Dissemination, Proceedings of 1995 USENIX Technical Conference, Janvier 1995.

- [Wats97] Mark Watson, Intelligent Java Applications for the Internet and Intranets, Morgan Kaufmann, 1997, 377 pages.

- [Wiel92] B.J. Wielinga, A.Th. Schreiber, J.A. Breuker, KADS: A Modelling Approach to Knowledge Engineering, Knowledge Acquisition, 1992.

- [Zobe98] J. Zobel, A. Moffat, Exploring the Similarity Space, SIGIR Forum, 32, pp. 18-34.