Università degli Studi di Roma Tor Vergata

Facoltà di Ingegneria

Corso di Modelli per la Gestione e la Ricerca dell'Informazione

TaskSpider

Progetto Web Spidering

Professore:
Roberto Basili

Studenti:
Simone Notargiacomo
Giuseppe Schipani

Anno Accademico 2007-2008

Contents

Chapter 1

Introduzione

Il *Web Spidering* è un campo molto importante nel mondo del Software e non solo, infatti tra le società più importanti vi è proprio $Google^{TM}$, che ha come core-business il settore dei motori di ricerca. Guardando alle moderne necessità informatiche, risulta evidente come lo *spidering* è e sarà fondamentale in tanti settori dell'industria e della ricerca. Questi motivi ci hanno spinto alla scelta del progetto riguardante il Web Spidering. La traccia richiede come obiettivi fondamentali la realizzazione di uno Web spider per informazioni tematiche, in particolare lo spider dovrebbe esplorare la rete filtrando tutte le pagine che non riguardano il tema scelto e memorizzare le rimanenti; inoltre dovrebbe essere possibile effettuare una ricerca sulle pagine ottenute. In tale trattazione verranno spiegate tutte le funzionalità del software realizzato, nonche la sua architettura e le varie scelte implementative. Infine verrà riportata anche la fase di testing con le relative conclusioni.

Chapter 2

Struttura progetto

Il progetto è stato strutturato seguendo un approccio simile all'*MVC*[1], in particolare è stata realizzata un'architettura suddivisa in tre livelli principali:

View: Riguarda la parte che interagisce con l'utente, ovvero l'interfaccia grafica e l'interfaccia web.

Controller: Questo livello contiene tutte le classi che vengono utilizzate per consentire al livello view di richiamare le funzionalità del software.

Model: Quest'ultimo livello contiene il cuore del progetto, che a sua volta può essere suddiviso in vari sottolivelli.

2.1 Model

Tale livello contiene le due parti fondamentali del software realizzato, che sono lo *spider* e il modulo di *Retrieval*. Il tutto è stato realizzato mantenendo la più alta modularità ed astrazione possibile, in modo da rendere semplice la manutenzione.

[1] Model-View-Controller

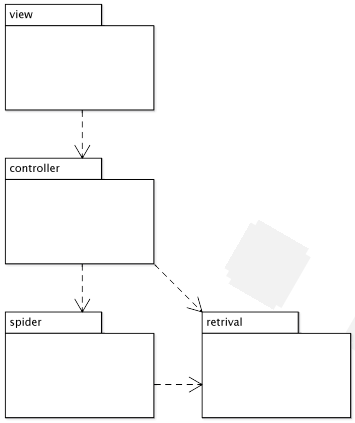

Figure 2.1: Architettura

2.1.1 Spider

Intuitivamente lo Spider potrebbe sembrare la parte più complessa del software, in realtà vi si è stato speso poco tempo rispetto al resto del progetto; questo perchè è stato utilizzato uno Spider già esistente, in particolare *Websphinx*[2]. Il lavoro speso sullo Spider si è incentrato sulla sua documentazione, e sul realizzare le nostre API di più alto livello, per una semplice gestione dello Spider. Inoltre si è fatto in modo che lo Spider si comporti come *Thread* consentendo al resto dell'applicazione di continuare ad operare, evitando delle lunghe attese dovute alla latenza dello spidering. Realizzando quanto appena detto è possibile interagire con lo Spider anche durante lo spidering, il che permettere di effettuare delle richieste tra cui: il numero di pagine recuperate, le pagine recuperate, interrompere lo spidering e avviare una nuova scansione.

SpiderExplorer

Come precedentemente detto è possibile recuperare le pagine esplorate, per realizzare tale lavoro è stato progettato un modulo chiamato *SpiderExplorer* che consente la navigazione dei risultati ottenuti dalla ricerca di quelli che si

[2] `http://www.cs.cmu.edu/$\sim$rcm/websphinx/`

attengono al Task richiesto[3]. Tale modulo effettua inoltre un operazione di indicizzazione sulle pagine, sfruttando il modulo di *Retrieval*.

2.1.2 Retrieval

Questo modulo consente la gestione del sistema di indicizzazione e di Retrieval di *Lucene*, infatti rende possibile indicizzare nuovi documenti (nel nostro caso informazioni sulle pagine recuperate) ed effettuare delle ricerche sui documenti presenti nell'indice. Questo come già accennato viene sfruttato dallo Spider per filtrare le pagine che si attengono al task da quelle che non sono rilevanti. Inoltre sono state aggiunte delle funzionalità che consentono l'espansione delle query tramite Rocchio e Wordnet. In seguito verranno discussi approfonditamente i moduli elencati.

2.2 Controller

Il Controller consente di gestire tutte le funzionalità dello Spider e del modulo di Retrieval, in particolare sarebbe un livello di astrazione più elevato che consente al livello successivo, ovvero View, di gestire tutto il software tramite l'ausilio di qualche semplice istruzione.

2.3 View

Questo layer potrebbe sembrare non molto rilevante, invece è risultato essere uno dei più complessi da realizzare. Tale difficoltà si è incontrata a causa della necessaria interazione tra l'applicazione desktop e l'applicazione web. Infatti per consentire una comoda navigazione dei risultati delle ricerche sono state progettate due interfacce grafiche, una web ed una desktop. La prima è molto più comoda per navigare tra i risultati all'interno del task, ad esempio con task = "Tor Vergata" ed una query = "Ingegneria". L'interfaccia desktop

[3]Un task potrebbe essere "Università"

invece è stata strutturata per consentire una semplice ed intuitiva navigazione delle pagine esplorate dallo Spider che si attengono al task, il tutto grazie ad un grafo esplorabile con il mouse.

Chapter 3

Spider

Come anticipato, il cuore dello Spider è stato preso da un progetto già esistente (*Websphinx*) per evitare di perdere molto tempo nella realizzazione da zero. Per spiegare meglio il funzionamento del sistema di spidering che abbiamo realizzato è riportato di seguito un esempio di funzionamento completo.

3.1 Funzionamento

Per avviare lo Spider è necessario indicargli un set di indirizzi di partenza, i quali possono essere ottenuti anche automaticamente tramite un modulo che è stato realizzato appositamente (vedere 3.2). Lo spidering può essere effettuato a qualsiasi livello di profondità, per evitare tempi di latenza troppo elevati si è deciso di impostare un range di 1-5, e come livello di default 3; con tale valore impostato è risultato uno spidering abbastanza veloce e i dati ottenuti rilevati. Altri parametri impostati nello spider sono:

1. il numero di Thread utilizzato nello spidering sono 8;
2. la dimensione massima di una pagina è 300kb;
3. abilitato l'utilizzo del file *robots.txt*;

4. politica di scansione delle pagine BFS (o DFS in base alla configurazione scelta dall'utente).

Il funzionamento generale dello spider è riassunto nei passi seguenti:

1. Impostate le varie opzioni iniziali è possibile avviare lo Spider che inizierà ad esplorare ogni pagina web a partire dal primo indirizzo del set di partenza;

2. durante la navigazione delle pagine, ad intervalli di tempo regolari viene avviato un secondo Thread che si occupa della scansione delle pagine esplorate per filtrare le pagine rilevanti dalle altre;

   (a) per ogni pagina rilevante si memorizzano tutte le sue informazioni che vengono utilizzate dal motore di Retrieval per l'indicizzazione; per avere dettagli vedere **??**;

   (b) viene controlla la presenza della pagina nell'indice, e se presente si confrontano le date delle due pagine per sapere se è necessario sostituire la pagina nell'indice con una più aggiornata;

3. lo spidering potrebbe andare avanti all'infinito se non fosse per il sistema che si è adottato per interromperlo; tale sistema consiste nel verificare, ad intervalli di tempo regolari, se il numero di pagine recuperate al precedente controllo è uguale al numero di pagine recuperate all'ultimo controllo. Questa operazione viene eseguita per circa 10 volte, il che rende abbastanza affidabile l'interruzione dello spidering.

3.2 Indirizzi automatici

Si è realizzato un modulo in grado di reperire degli indirizzi di partenza per lo spidering. Tale modulo è stato progettato sfruttando le API di Google^{TM}, le quali ci consentono di effettuare una ricerca sul web tramite il suo motore di ricerca. In questo modo si può avviare lo Spider con un set di indirizzi molto rilevanti il che ci consente di avere dei risultati validi. L'unico problema che si

è riscontrato con l'utilizzo di tali API è che siamo stati costretti ad utilizzare un servizio che Google^{TM} fornisce solo per mantenere la compatibilità con vecchie applicazioni, infatti ultimamente Google^{TM} ha introdotto dei servizi sostitutivi che consentono di effettuare tali ricerche solo da pagine web e non più da codice *Java*.

Chapter 4

Retrieval

Una parte fondamentale del progetto è proprio quella riguardante il *Retrieval*, che gestisce tutte le operazioni riguardanti l'indice dei documenti e la ricerca. In particolare si è realizzato un sistema che sfrutta la libreria *Lucene* per gestire in modo semplice l'indice.

4.1 Documenti

Il software che si è realizzato si basa fortemente sulla gestione dei documenti registrati nell'indice, quindi risulta fondamentale riportare la struttura dei documenti. Ogni documento ha un riferimento uno ad uno con una pagina web recuperata, infatti vengono salvati ed indicizzati i seguenti campi:

URL: tale campo contiene semplicemente l'*URL* da cui è stato possibile raggiungere la pagina in questione;

Title: contiene il titolo della pagina;

Description: alcune pagine web hanno dei tag speciali che contengono la descrizione del contenuto della pagina;

Keywords: le parole chiavi contenute nel tag keywords;

Keyphrases: simili alle keywords, ma più rare;

Body: contiene il contenuto del tag body (filtrato), ovvero gran parte della pagina recuperata;

Date: sarebbe la data di scadenza della pagina, viene utilizzata per aggiornare un documento nell'indice in caso se ne reperisca uno più recente;

Tutti i campi elencati sono necessari per poter effettuare una ricerca sufficientemente rilevante per la portata del progetto. Come può essere facilmente intuito, il campo più importante risulta essere *Body*, oltre all'indispensabile *URL* che viene usato come chiave per il documento; infatti non possono coesistere nell'indice più documenti con lo stesso *URL*. Ogni documento che viene inserito nell'indice deve passare una serie di controlli di integrità. Nel caso del body vengono effettuate delle operazioni di pulitura del testo; in particolare vengono rimosse tutte le parole che riguardano codice o quant'altro. Grazie a quest'ultima operazione si ottiene un indice molto più pulito e rilevante per le ricerche.

4.2 Indicizzatore

Un'altra parte importante del software in questione è l'*Indicizzatore* (o *Indexer*) che si occupa della gestione vera e propria dell'indice dei documenti. In particolare, l'Indexer consente di aggiungere dei documenti nell'indice, evitando l'inserimento di duplicati (vedere 4.1); oltretutto è anche in grado di sfruttare il campo *Date* per un eventuale aggiornamento del documento. L'Indexer viene chiamato ad intervalli di tempo regolari per consentirgli di indicizzare tutti i documenti rilevanti trovati. Questo modulo è risultato essere uan parte del progetto in cui viene speso gran parte del tempo, a causa dell'elevato numero di nuovi documenti da aggiungere (o modificare) nell'indice.

4.3 Ricerca

La ricerca è un altro tassello fondamentale per il raggiungimento degli obiettivi del progetto; infatti la ricerca è necessaria sia per filtrare i documenti in base al task e sia effettuare delle ricerche sui documenti filtrati. Per tali motivi sono stati realizzati due moduli, uno per ogni tipo di ricerca.

4.3.1 Ricerca task

Il primo tipo di ricerca genera una query a partire dal task richiesto, in particolare vengono espletati i seguenti passi:

1. si prende il task inserito dall'utente, che potrebbe essere "Università Tor Vergata", e lo si divide in token (Università, Tor, Vergata);
2. per ogni campo ("keywords", "title", "body") si prepara una query del seguente tipo:

   ```
   title:Università AND title:Tor AND title:Vergata
   ```

3. in seguito si mettono in OR tutte le query preparate per ottenere:

   ```
   (title:Università AND title:Tor AND title:Vergata) OR
                        (...)  OR (...)
   ```

Formulando tale query non si fa altro che ricercare nell'indice uno o più documenti con i parametri richiesti. Si deciso di imporre il vincolo che l'intero task deve essere presente in almeno un campo, questo per rendere più rilevanti i risultati della ricerca. In principio si erano impostati altri tipi di vincoli, in particolare era sufficiente che un solo token fosse presente in un qualsiasi campo per rendere il documento rilevante; il che rendeva un set di risultati molto scadenti.

4.3.2 Ricerca query (Wordnet - Rocchio)

Per quando riguarda la ricerca all'interno dei documenti già filtrati tramite "ricerca task", si è deciso di implementare un sistema molto più complesso ed efficace. Il testo di ricerca che viene inserito dall'utente può risultare poco specifico, il che rende i risultati molto vaghi; per evitare di incorrere in tali problemi si è deciso di implementare un sistema di *Query Expansion*. In principio si è realizzato un modulo di espansione basato su *Wordnet*, in seguito se ne è aggiunto uno basato su *Rocchio* che ha avuto un impatto migliore sui risultati.

```
isearcher = new IndexSearcher(indexDir);
queryString = queryString.replaceAll("%20", " ");

StandardAnalyzer analyzer = new StandardAnalyzer();
QueryParser parser = new QueryParser("body", analyzer);
Query query = parser.parse(queryString);

Debug.println("Normal: "+query.toString(), 1);
result = isearcher.search(query);
Debug.println("Search hits: "+result.length(), 1);

Query expandedQuery = this.expandQuery(query, query.toString(), result,
    isearcher, analyzer, type);
Debug.println("Expanded: "+expandedQuery.toString(), 1);
result = isearcher.search(expandedQuery);
Debug.println("Search with expanded query hits: "+result.length(), 1);
```

Il funzionamento della ricerca nell indice è riportato di seguito:

1. viene inizializzato il modulo di ricerca nell'indice;
2. eventualmente si pulisce la stringa di query che potrebbe contenere caratteri speciali come %20 , dovuti alla pagina jsp;
3. si avvia la ricerca con la query ricevuta;
4. in caso di Rocchio i risultati ottenuti vengono utilizzati per l'espansione della query. In caso di Wordnet, invece, si espande chiamando il modulo

adeguato;

5. si effettua la ricerca con la nuova query espansa;

Wordnet Tale componente è formato da:

1. un file, scritto in *Prolog*, contenente tutte le parole di *Wordnet*;
2. un modulo in grado di indicizzare le parole in un indice tramite *Lucene*;
3. un altro modulo che ricerca tutti i sinonimi delle parola che si richiede.

Utilizzando questo sistema si è raggiunta una precisione nei risultati più elevata. In alcuni casi si incombe in ambiguità a causa del constesto in cui si trovano le parole, ad esempio se si espande una query del tipo "Apple" il sistema aggiunge parole del tipo "fruit", il che non sarebbe sbagliato se noi stessimo cercando il frutto; ma se si volesse ricercare la società "Apple" allora il risultato di Wordnet sarebbe totalmente errato.

Rocchio Per evitare di incorrere in problemi come quelli appena citati si è realizzato un altro modulo di espansione query utilizzando il sistema di *Pseudo-relevance feedback Rocchio*. Tale metodo funziona nel seguente modo:

1. si prende la query desiderata dall'utente e si effettua la ricerca nell'indice;
2. con i risultati ottenuti si effettua lo pseudo-relevance feedback considerando rilevanti i primi N documenti ed irrilevanti gli ultimi M;
3. si prendono i termini dei documenti rilevanti e si aggiungono alla query originale;
4. si sottraggono i termini dei documenti non rilevanti;
5. viene rieseguita una nuova ricerca con la nuova query.

La formula utilizzata per l'espansione è la seguente:

$$\overrightarrow{q}_m = \alpha \overrightarrow{q} + \frac{\beta}{|D_r|} \sum_{\forall \overrightarrow{d_j} \in D_r} \overrightarrow{d_j} - \frac{\gamma}{|D_n|} \sum_{\forall \overrightarrow{d_j} \in D_n} \overrightarrow{d_j}$$

E' stata implementata una variante di *Rocchio* aggiungendo ai nuovi termini della query il loro peso *tf-idf*; in particolare la formula è diventata:

$$\overrightarrow{q}_m = \alpha \overrightarrow{q} + \frac{\beta}{|D_r|} \sum_{\forall \overrightarrow{d_j} \in D_r} \overrightarrow{w_j} \overrightarrow{d_j} - \frac{\gamma}{|D_n|} \sum_{\forall \overrightarrow{d_j} \in D_n} \overrightarrow{w_j} \overrightarrow{d_j}$$

in cui il simbolo $\overrightarrow{w_j}$ indica il vettore dei pesi *tf-idf* dei termini nel documento $\overrightarrow{d_j}$. Tale funzionalità è risultata poco rilevante ai fini della precisione considerando che *Lucene* utilizza già tale pesatura nella ricerca dell'indice, quindi questa variante può essere disabilitata dall'utente in base alle sue preferenze. Il metodo *Rocchio* (senza variante) ha consentito un aumento della precisione nei documenti ritrovati, infatti a differenza del metodo *Wordnet* è molto più difficile incorrere in ambiguità delle parole. Un punto a favore di *Wordnet* può andare sotto l'aspetto della espressività della query, questo perchè con il sistema *Rocchio* realizzato non possono essere fatte espansioni di query che specificano il campo ("URL", "title" ecc.) del termine. Per evitare un'elevata complessità computazionale è stata impostata una espansione che considera soltanto il campo "body". Tale vincolo ha senso anche sotto altri aspetti, come il fatto che gran parte del contenuto informativo di una pagina web è proprio nel "body".

Chapter 5

View

In questo capitolo si tratterà la componente *View* del sistema realizzato. In tale livello è stata implementata l'interfaccia grafica che consente una semplice e rapida interazione con lo Spider, in particolare le interfaccie progettate sono di tipo:

Desktop Application: realizzata puramente per utilizzo di tipo desktop;

Web Application: per un utilizzo di tipo web, ovvero tramite browser.

Si è deciso di intraprendere questa strada per rendere il più user-friendly possibile la nostra applicazione, infatti, grazie alla capacità di gestione delle interazioni desktop⇔web che ha il Java, è risultato molto semplice comunicare i risultati delle ricerche da desktopApp alla webApp. In questo modo si è realizzata una interfaccia molto simile ai motori di ricerca che vengono utilizzati quotidianamente sul web. Un esempio pratico di quanto esposto può essere il seguente:

1. si inseriscono il task e la query;
2. si avvia lo spidering;
3. ad intervalli viene effettuata una ricerca della query specificata;
4. i risultati vengono visualizzati su una pagina web (in particolare jsp);

5. l'utente è in grado di navigare i risultati per pagine, come fossero i risultati di un motore di ricerca stile Google^{TM}.

5.1 Desktop Application

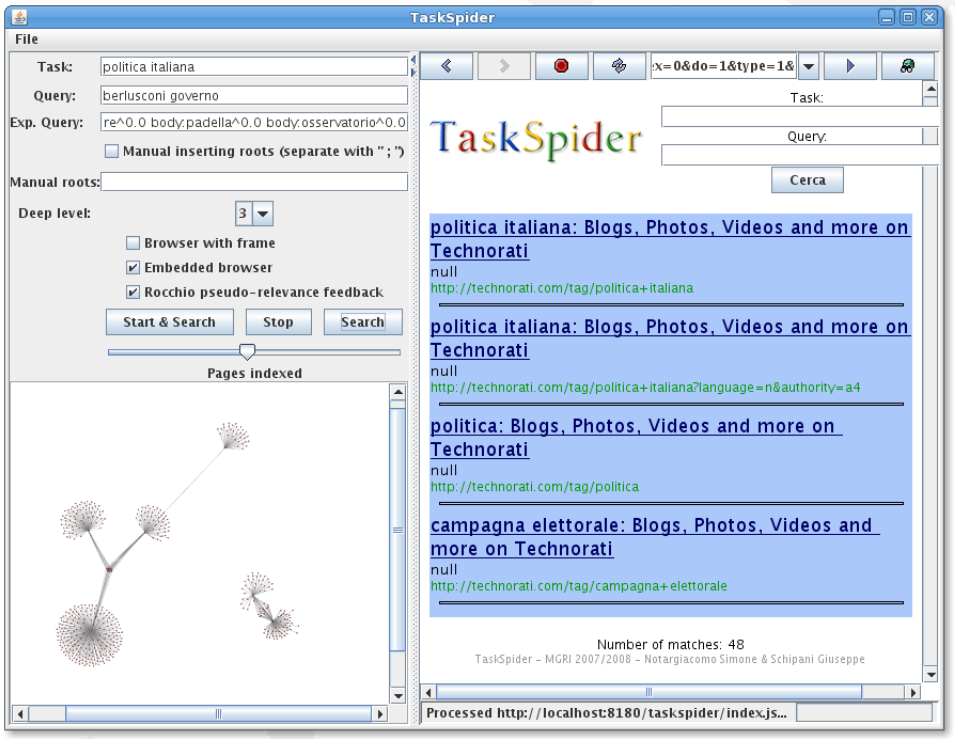

Figure 5.1: Desktop Application

L'interfaccia grafica realizzata per desktop è formata da:

- alcuni campi per l'immissione del task e della query;
- campi di opzioni per scegliere le modalità di ricerca, visualizzazione ed altro;
- pulsanti per avviare lo spidering e la ricerca;
- un grafo rappresentante la rete delle pagine web filtrate;

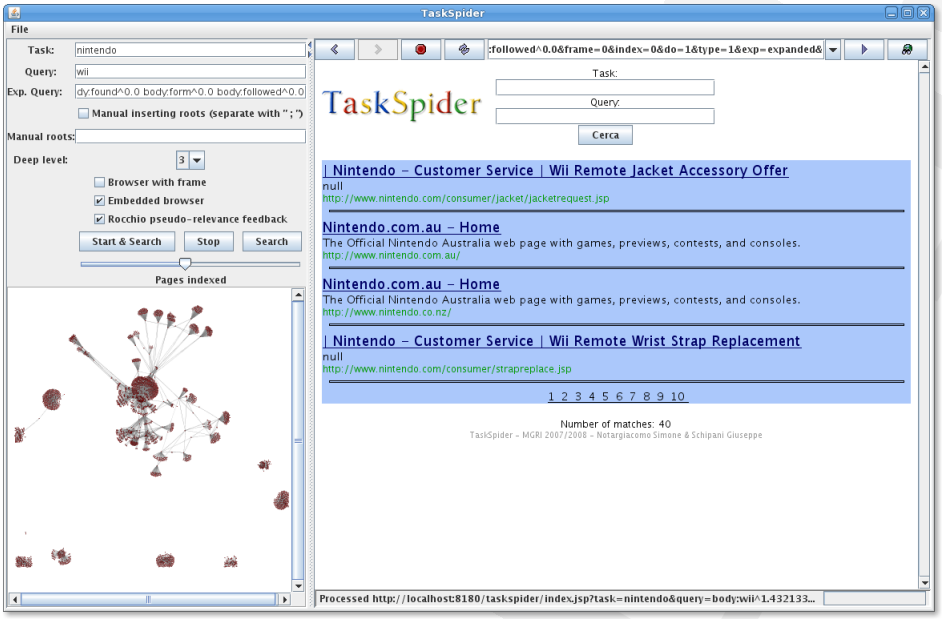

Figure 5.2: Desktop Application

- un browser web per visualizzare i risultati della ricerca e navigarli.

5.2 Web Application

Si è deciso di realizzare una pagina web in grado di effettuare delle ricerca sugli indici e in grado di rappresentare i risultati sotto varie forme. In particolare è stata implementata considerando due modalità di utilizzo:

Solo ricerca Questo caso si ha quando si accede alla pagina direttamente dal browser, senza passare argomenti tramite barra degli indirizzi. Questa modalità consente di effettuare una ricerca sugli indici già memorizzati sul sistema; la ricerca può essere avviata utilizzando i campi Task e Query appositamente creati. I risultati possono essere visualizzati sia come un semplice elenco di indirizzi oppure anche con un albero che riunisce i risultati per dominio.

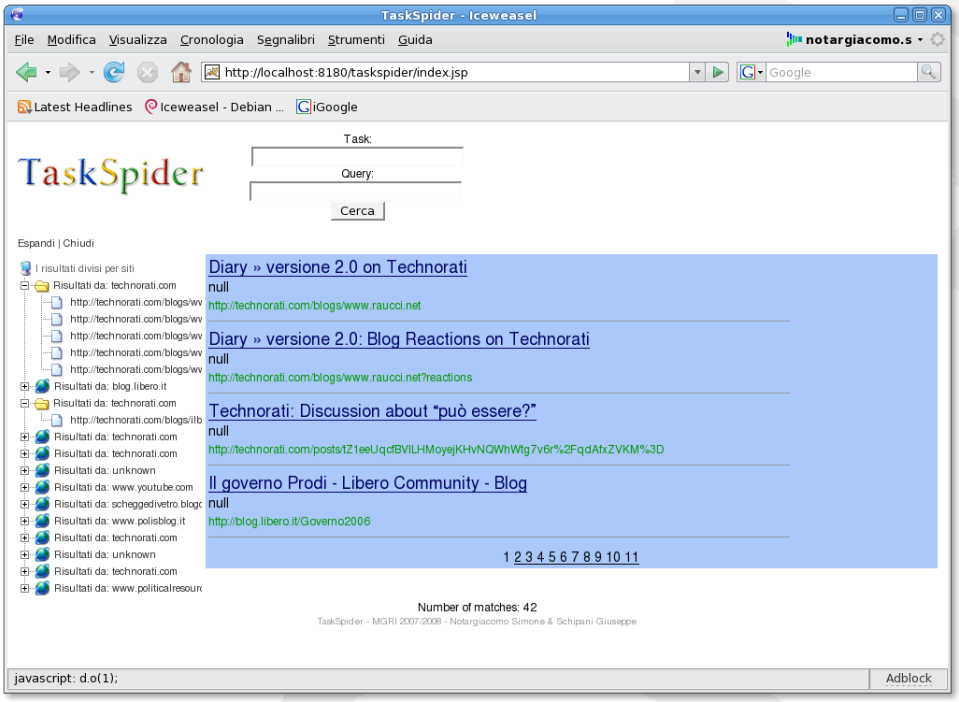

Figure 5.3: Web Application

Spidering e Ricerca Questo secondo caso si verifica quando la pagina web viene acceduta dalla Desktop Application, la quale effettua lo spidering ed in seguito richiama la pagina. E' opportuno specificare che la pagina, in realtà si comporta sempre nello stesso modo, ovvero esegue una ricerca sull'indice richiesto. C'è un unica differenza rispetto al caso precedente, ovvero che nella modalità attuale tutti i dati per effettuare la ricerca vengono passati tramite barra degli indirizzi (*QueryString*); inoltre l'albero dei risultati per domino è opzionale.

Chapter 6

Performance

La fase finale dello sviluppo di tale progetto è stata la valutazione delle perfomance effettuando i relativi test. In particolare si è deciso di effettuare vari test utilizzando varie combinazioni dei parametri $\alpha\beta\gamma$. Inoltre sono stati utilizzati due meccanismi di *Query Expansion*: *Rocchio Pseudo-Relevance-Feedback* e *Wordnet*. Per lo spidering delle pagine si è utilizzato sia il metodo manuale di inserimento indirizzi di partenza e sia il metodo automatico (vedere 3). Con il metodo automatico si è riscontrato un elevato aumento della precisione, questo perchè si inizia lo spidering da pagine che hanno un'alta rilevanza al task. Di seguito sono riportati i vari test con i relativi valori.

6.1 Test 1

La prima prova è stata effettuata sul task *music*, lasciando agire il crawler per circa un ora dopodichè sono state fatte le query. Con un task di questo tipo sono state indicizzate oltre 3000 pagine web, il che rende la ricerca abbastanza veritiera. I link iniziali sono stati specificati a mano, ovvero: `http://www.mtv.com`, `http://www.myspace.com`; il che ha reso la precisione più bassa rispetto all'utilizzo degli indirizzi automatici.

N	α	β	γ	DocRel	DocNonRel	Rocchio	Wordnet	Normale
1	1	1	0.5	10	4	0.41	0.27	0.27
2	1	1	0	10	4	0.30	0.27	0.27
3	1	1	0.5	10	10	0.52	0.27	0.27
4	1	1	0	10	10	0.38	0.27	0.27

Table 6.1: Test 1

Dalla tabella si può notare che la precisione di *Wordnet* e del metodo normale sono uguali, questo è dovuto al fatto che *Wordnet* in alcuni casi non riesce ad espandere perchè non conosce il termine.

6.2 Test 2

La seconda prova è stata effettuata sul task *politica italiana*, lasciando agire il crawler per circa un ora dopodichè sono state fatte le query. Con un task di questo tipo sono state indicizzate oltre 4000 pagine web.

N	α	β	γ	DocRel	DocNonRel	Rocchio	Wordnet	Normale
1	1	1	0.5	10	4	0.45	0.55	0.55
2	1	1	0	10	4	0.33	0.55	0.55
3	1	1	0.5	10	10	0.56	0.55	0.55
4	1	1	0	10	10	0.41	0.55	0.55

Table 6.2: Test 2

Tutti questi test sono stati eseguiti anche con la variante di *Rocchio* attivata ed i risultati sono stati più scadendi, infatti si sono ottenuti dei valori di precision più bassi di circa il 10%.

Chapter 7

Conclusioni

Dai vari test effettuati è stato rilevato che per avere una precisione abbastanza alta si deve effettuare lo spidering su una grande mole di pagine, ed inolte è necessario l'utilizzo di *Rocchio* come metodo di espansione delle query; inoltre utilizzando il metodo automatico di recupero degli indirizzi iniziali si ha un incremento della precisione abbastanza sensibile. Con l'utilizzo del metodo *Wordnet* si sono ottenuti valori più bassi nella maggior parte dei casi dovuti all'ambiguità delle parole. Una possibile miglioria da apportare al progetto sarebbe sicuramente il metodo di indicizzazione, il quale potrebbe essere migliorato a livello di prestazioni dell'algoritmo. Un'ulteriore migliora da apportare riguarderebbe la ricerca nei documenti indicizzati, in particolare si potrebbe sfruttare il metodo *LSI* per l'indicizzazione e la ricerca, il che migliorerebbe ulteriormente la rilevanza dei risultati.

List of Tables

List of Figures

Index