UNIVERSITÀ DEGLI STUDI DI ROMA
TOR VERGATA

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Un ambiente Web per l'indicizzazione ed il recupero multilinguale di documenti multimediali

Relatore
Prof. Roberto Basili

Candidato
Andrea Vanzo

Co-relatore
Ing. Diego De Cao

Anno Accademico 2008/2009

Indice

Elenco delle figure

1 Introduzione

1.1 Obiettivi

Da qualche tempo, il problema del recupero dell'informazione è diventato il fulcro degli studi di un ramo dell'informatica. Anche la parola "informatica", nel suo significato [dal francese "*informatique*", ovvero INFORMazione automATICA] sta ad indicare l'analisi delle tecniche algoritmiche che descrivono e manipolano l'informazione.

In una società che da la possibilità a chiunque di accedere ad una grandissima quantità di informazioni, il problema generale della *ricerca* è diventato di estrema attualità. Inoltre, lo sviluppo del Web e la sua grande espansione, ha fatto si che molti studi si concentrassero sul problema del recupero dell'informazione, sullo sviluppo di tecniche sempre più efficienti e che dessero risultati sempre più accurati e consistenti con il bisogno dell'utente.

La mia tesi mira proprio allo studio dell'*Information Retrieval* [1], delle tecniche utilizzate, degli strumenti sviluppati a supporto degli utenti e dei programmatori, con l'obiettivo di poter utilizzare suddette tecniche per la progettazione e l'implementazione di un *search engine* destinato ad una particolare classe di contenuti multimediali che analizzeremo più avanti.

Affronteremo l'*Information Retrieval* nel capitolo 2, facendo un ampia analisi dei metodi di indicizzazione più comunemente utilizzati, per poi trattare le architetture distribuite a servizio dell'IR. Parleremo dell'architettura *three-tier*, dei

tre strati che la compongono ed entreremo nello specifico analizzando gli strumenti utilizzati nell'implementazione della nostra applicazione. Si introdurrà inoltre il problema del CLIR (*Cross-Lingual Information Retrieval*) [2], ovvero la possibilità di sviluppare strutture di IR che siano portabili a livello di linguaggio.

Nel capitolo 3 verrà presentato il mio lavoro, partendo dalle specifiche, dai problemi riscontrati, fino ad arrivare alle soluzioni strutturali trovate ed alle applicazioni implementate.

Con il capitolo 4 proveremo ad analizzare i possibili sviluppi futuri del progetto.

Nell'Appendice sono presenti alcune tipiche sessioni utente.

1.2 Problematiche nel recupero delle informazioni multimediali

Uno dei problemi principali dell'IR, sul quale ci concentreremo, è quello delle differenze nei domini di ricerca. Come sappiamo, infatti, una collezione può essere formata da materiale eterogeneo, in cui ogni documento viene attribuito ad una delle seguenti classi:

- **testuale**
- **immagini**
- **audio**
- **audio/video**

che vengono rappresentati e manipolati in maniera differente. Un'immagine, per esempio, può essere come una matrice di pixel, uno spazio vettoriale, rappresentazioni che sono abbastanza lontane da quella testuale, o da quella audio.

Il caso più semplice è quello in cui i documenti che dobbiamo indicizzare appartengono tutti al dominio testuale. Infatti, per ottenere i risultati desiderati, basterà andare alla ricerca delle occorrenze dei termini della *query* (la stringa digitata dall'utente che rappresenta il suo *information need* [3]) nei documenti della collezione e restituire i risultati.

Ma in che modo è possibile fare *retrieval* di un documento audio/video, utilizzando una *query* che, come sappiamo, viene espressa con una forma testuale?

I *metadati* sono una soluzione che permette di identificare un documento e fornire informazioni circa il suo contenuto.

Prendendo come esempio un'immagine, i *metadati* da utilizzare per la sua indicizzazione potrebbero essere:

- autore
- nome dell'immagine
- data di creazione
- soggetto
- location
- …

A differenza di un documento testuale, in cui i termini formano il proprio il corpus del documento, nel caso dell'immagine, i *metadati* non "sono" il documento, ma lo rappresentano grazie alle informazioni che contengono.

È così possibile recuperare anche documenti multimediali, andando ad insistere sul contenuto dei *metadati* e sulla rappresentazione che essi danno del documento.

Un discorso analogo si può fare anche per oggetti audio e audio/video, ovviamente considerando *metadati* di interesse opportuni.

Nel caso, per esempio, di uno show televisivo, potremo utilizzare i *metadati* riguardanti il canale di trasmissione, il titolo del programma, l'orario di trasmissione, l'edizione, il conduttore,... in modo da caratterizzare nella maniera più completa possibile il contenuto del documento.

2 Information Retrieval

2.1 Introduzione

"L'Information Retrieval è l'insieme delle tecniche che consentono di recuperare materiale non strutturato da una grande quantità di dati, soddisfacendo delle informazioni desiderate" [3]

Questa è solo una delle tante definizioni di *Information Retrieval*, ovvero *recupero dell'informazione*.

Per capire meglio l'importanza dell'IR, basti pensare ad alcune semplici operazioni nella vita di tutti i giorni: il magazziniere che consulta il terminale per sapere in quale settore ed in quale scaffale risiede l'oggetto desiderato, il bibliotecario che cerca il libro richiesto dal cliente, …

L'IR, però, non riguarda solamente categorie di persone che ricoprono particolari ruoli: infatti, con lo sviluppo del Web e l'espansione della sua fruibilità a gran parte della popolazione mondiale, l'IR è diventato un problema ancora più attuale e di fondamentale importanza. Chiunque abbia avuto la possibilità di accedere al Web, si è trovato per forza di cose a doversi servire di un motore di ricerca per recuperare le informazioni desiderate; e quante volte capita di dover effettuare più di una ricerca, provare termini diversi per arrivare ai risultati voluti.

L'obiettivo dell'IR è proprio questo, quello di sviluppare tecniche avanzate per cercare di ridurre il più possibile il *gap* tra la *query* digitata (che rappresenta l'*information need* dell'utente) e l'insieme delle risposte restituite dal *search*

engine.

Alla luce di quanto detto, andremo ora ad analizzare in modo più approfondito alcune delle varie tecniche e aspetti dell'IR, partendo dal più semplice e meno efficiente *Boolean Retrieval*, fino ad arrivare ai metodi più complessi di *Ranked Retrieval*.

2.2 Modelli di *Information Retrieval*: Boolean e Vector Spaces

Come abbiamo già anticipato nell'introduzione al capitolo, esistono vari modelli che implementano il concetto di *retrieval*, ove per modello si intende una particolare rappresentazione delle strutture dati e dei processi di *retrieval*.

Il primo di cui parleremo è il *Boolean Retrieval Model*. L'idea principale è quella di utilizzare gli operatori booleani nella costruzione della *query* e consentire le operazioni di ricerca nei documenti in base all'occorrenza o meno dei termini.

Come è facile capire, questo è il modello più semplice di *retrieval* e, ovviamente, il meno efficiente, a causa della scarsa elasticità sia nella formulazione della *query*, sia nella generazione dei risultati trovati.

Il *Vector Space Model* [4], invece, si pone nel campo dei modelli di *ranked retrieval*, più accurati e complessi, in cui le strutture dedicate alla persistenza dei dati vengono arricchite di informazioni. I concetti di pesatura e di similarità, permettono di assegnare punteggi sia ai termini, in fase di *querying*, che ai risultati in fase di *retrieval*. Questo fa si che documenti siano più consistenti di altri con l'*information need*, assicurando perciò maggiore flessibilità alla ricerca.

2.2.1 Boolean queries, l'approccio "booleano" agli inverted indexes

Immaginate che si debba determinare quali opere di Giacomo Leopardi contengono le parole **luna AND triste AND NOT mortale.** Un modo per poter dare una risposta alla *query* è quello di scansionare tutti i documenti, uno ad uno, cercando in ognuno di questi le occorrenze dei termini *luna*, *triste* e *mortale*. Nel nostro caso questa potrebbe non essere una operazione molto dispendiosa in termini di tempo di ricerca e di complessità dell'algoritmo, ma in altre situazioni potremmo ottenere di più:

- scandire una vasta collezione di documenti in modo veloce;

- permettere *matching* più flessibili (con una *query* della forma di quella sopra citata, sarebbe impensabile esprimere la ricerca di termini vicini, come per esempio **silenziosa NEAR luna**);
- permettere di poter ottenere risultati ordinati in base alla loro rilevanza, ovvero alla loro attinenza alla *query*.

Un modo per risolvere il problema della scansione lineare dei testi a *run-time* è quello di *indicizzare* la collezione; questa è un'operazione preliminare e viene effettuata come "preparazione" alla ricerca.

Supponiamo ora di costruire una matrice in cui abbiamo in una dimensione tutte le parole usate da Leopardi nelle sue opere e nell'altra dimensione tutte le opere dello scrittore. All'interno della matrice, nella cella corrispondente alla parola *i* e opera *j*, avremo 1 se il termine *i*-esimo è contenuto nel documento *j*-esimo, mentre avremo 0 altrimenti.

Il risultato sarà una matrice della forma:

	Il passero solitario	*L'infinito*	*Canto notturno di un pastore errante dell'Asia*	*A Silvia*	*Il sabato del villaggio*	*Alla luna*
luna	0	0	**1**	0	**1**	**1**
triste	0	0	0	0	0	**1**
mortale	0	0	**1**	**1**	0	0
quiete	0	**1**	0	**1**	0	0
...						

Figura 2.1 *- Term-document incidence matrix*

chiamata *matrice di incidenza termine-documento* [3].

Ogni termine sarà perciò rappresentato da un vettore che ci indica l'occorrenza o meno della parola nel documento. Ma anche ogni opera sarà rappresentata da un vettore che ci indica tutte le parole che occorrono nel documento.

Alla luce di quanto detto possiamo perciò rispondere alla *query* iniziale riscrivendoci ogni termine come vettore binario e facendo un confronto bit a bit:

001011 AND 000001 AND NOT 001100 =

001011 AND 000001 AND 110011 = 000001

La risposta alla *query* è Alla luna.

Il *Boolean retrieval model* è proprio questo: un modello di recupero delle informazioni nel quale possiamo esprimere la nostra *query* semplicemente utilizzando gli operatori booleani AND, OR e NOT.

Dopo aver introdotto il *Boolean retrieval model*, diamo ora delle definizioni che ci aiuteranno a capire meglio questo modello e quelli che affronteremo più avanti.

- **Documento**: per documento si intende qualsiasi contenuto informativo su cui abbiamo deciso di implementare un sistema di IR. Un documento può essere una poesia o una notizia, i capitoli di un libro piuttosto che un articolo scientifico.
- **Collezione**: rappresenta l'intero insieme dei documenti su cui vogliamo poter effettuare le ricerche.
- **Information need**: è l'argomento del quale l'utente desidera ottenere dei risultati o delle informazioni.
- **Query**: a differenza dell'*information need* la *query* rappresenta la stringa vera e propria che l'utente digita (per esempio, su un motore di ricerca) per comunicare il proprio *information need*.
- **Relevance**: un documento è rilevante se l'utente ne percepisce la consistenza con il suo *information need*.
- **Effectiveness**: è l'efficacia di un sistema di IR (la qualità dei risultati) e viene di solito valutata con due parametri:
    - *Precision*: quanti risultati sono consistenti con l'*information need*
    - *Recall*: quanti documenti rilevanti dell'intera collezione sono tra i risultati della *query*.

Per capire l'importanza degli *inverted indexes*, che tratteremo più avanti, nelle

tecniche di IR facciamo un esempio: supponiamo di avere una collezione di 1 milione di documenti e che ogni documento contenga in media 1000 parole. Assumendo una media di 6 bytes a parola, avremmo una quantità di dati di circa 6 GB. Considerando ora che la collezione ha in media 500000 termini distinti, otterremmo una matrice di 500K × 1M = 500000000000 *entries* di 0 o 1; poiché ogni documento ha circa 1000 parole, le uniche *entries* ad avere gli 1 (le informazioni rilevanti) sarebbero solamente non più di un miliardo, ovvero avremmo un minino del 99.8% di zeri [3].

Nasce allora proprio da qui la necessità di dover ridurre al minimo lo spreco di risorse.

L'idea di base è quella di tener traccia solamente della informazione necessaria, ovvero di quello 0.2% di 1 all'interno della matrice di incidenza: gli *inverted indexes*.

L'*inverted index* è costituito principalmente da due parti: il *dizionario*, che è l'insieme di tutti termini contenuti nella collezione, e la cosiddetta *posting list*, una per ogni termine, che è una lista che tiene traccia dei documenti nel quali il termine, a cui la lista fa riferimento, occorre *(fig. 2.2)*.

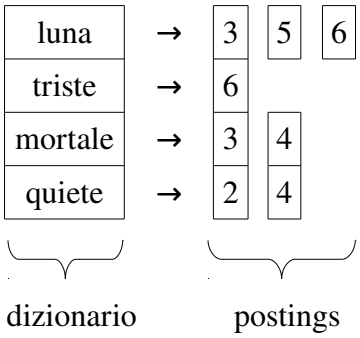


Figura 2.2 *- Inverted Index*

In questo caso abbiamo assunto che ogni documento abbia un proprio ID univoco che lo contraddistingue, in modo da poter puntare direttamente al documento stesso.

2.3 Inverted indexes

Abbiamo introdotto nel paragrafo precedente gli *inverted indexes* nella loro forma più semplice. Ovviamente, per cercare di ottimizzare le ricerche, sono stati

sviluppati indici con strutture più complesse, andando a migliorare sia i tempi di ricerca all'interno del vocabolario dei termini, sia la quantità di informazioni nelle *posting lists*.

La costruzione di un *inverted index* si basa sui seguenti passi:

- raccogliere tutti i documenti da indicizzare;
- suddividere ogni documento in *tokens* (per esempio, ogni parola rappresenta un *token*);
- normalizzare i *token* secondo le regole grammaticali della lingua che si vuole utilizzare (per esempio, in inglese, togliere tutte le *"s"* che rendono un sostantivo plurale);
- costruire l'*inverted index* associando ad ogni termine del dizionario una *posting list* con tutti i documenti in cui quel termine occorre;
- infine, passo non strettamente necessario, ordinare l'indice.

Questi *step* possono perciò essere suddivisi in due momenti cruciali: il momento della determinazione del vocabolario dei termini e quello della costruzione delle *posting lists*.

Diamo ora uno sguardo più approfondito alle strutture e alle tecniche più comuni nella realizzazione di un *inverted index*.

2.3.1 Determinare il vocabolario dei termini

Il primo passo nella indicizzazione di una collezione di documenti, è quello di determinare tutte le parole che saranno per noi rilevanti e che ci serviranno per costruire il vocabolario dei termini.

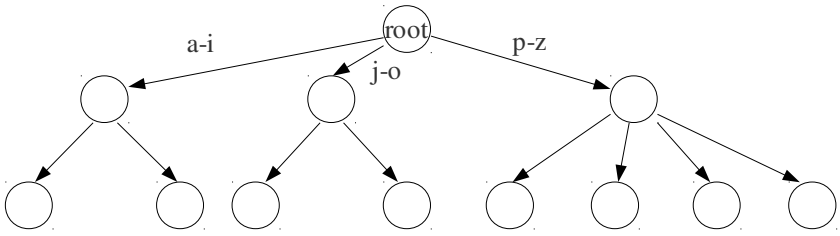


Figura 2.3 *- B-tree*

L'importanza di determinare un'ottima configurazione e struttura del vocabolario, deriva dal fatto che tutti i termini scelti saranno l'unica rappresentazione che abbiamo dei documenti e che una struttura ottimizzata per la

ricerca (es. *fig. 2.3* che mostra un vocabolario costruito con una struttura ad albero, in cui ogni nodo può avere un numero di figli compresi nell'intervallo [a, b]), permetterà di raggiungere i risultati in tempi minori [5].

Le principali tecniche nel processo di determinazione dei termini del vocabolario sono:

- *tokenization*
- *normalization*
- *stemming* e *lemmatization*

La *tokenization* è l'operazione preliminare che serve a suddividere il documento in *token*, ovvero unità semantiche indipendenti fra loro. La scelta dei *token* viene effettuata anche in base a quale politica verrà scelta per la gestione delle *stop words* (tutte quelle parole che ricorrono frequentemente nella collezione e che perciò non vengono considerate rilevanti ai fini della caratterizzazione del documento) [3].

La *normalization*, invece, ha come obiettivo quello di eliminare tutte quelle ambiguità che possono nascere dai vari modi in cui un termine può essere scritto (es. USA, U.S.A., United States of America,...) [3].

Stemming e *lemmatization* sono processi che servono a ricondurre i termini alla loro radice originale (es. democratico, democrazia... → democra*), come nel caso dello *stemming*, o al loro lemma originale (es. ho, avessi, avrebbe... → avere) come nella *lemmatization* [3].

2.3.2 Posting lists

Come per il vocabolario, anche le *posting lists* necessitano di particolari tecniche e strutture per la loro ottimizzazione.

Un esempio sono le *skip lists* [5], liste con puntatori che permettono di saltare alcuni step nella ricerca delle occorrenze nei documenti *(fig. 2.4)*.

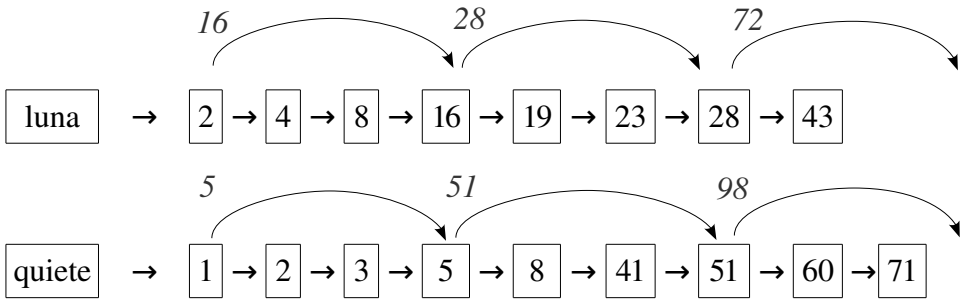


Figura 2.4 *- Skip lists* [5]

Nelle *positional posting lists* [3], invece, la lista dei documenti viene arricchita con informazioni riguardanti la posizione del relativo *token* all'interno del documento, la frequenza del termine nella collezione e nel singolo documento *(fig. 2.5)*.

il, 993427:
⟨1, 6: ⟨7, 18, 33, 72, 86, 231⟩;
2, 5: ⟨1, 17, 74, 222, 255⟩;
4, 5: ⟨8, 16, 190, 429, 433⟩;
5, 2: ⟨363, 367⟩;
7, 3: ⟨13, 23, 191⟩; ...⟩

Figura 2.5 *- Esempio di positional index dove il termine "**il**" ha una document frequency di 993427 e occorre 6 volte nel documento con id 1, nelle posizioni 7, 18, 33...*

2.4 Il *Vector Space Model* (VSM)

Il *Vector Space Model* è un modello di rappresentazione geometrica dei documenti introdotta da G. Salton [4]. Tale modello è quello maggiormente utilizzato nell'ambito dei sistemi di *retrieval* quali motori di ricerca, anche perché esso nasce proprio come tecnica finalizzata a task di ricerca di informazioni.

Questo modello di *ranked retrieval* estende il concetto di documento intrapreso nel paragrafo 2.2.1: come si può vedere nella *term-document incidence matrix (fig. 2.1)* ogni documento viene rappresentato attraverso un vettore a t dimensioni, dove t sono tutti i termini che occorrono nel documento.

VSM è una tecnica che prende in considerazione uno spazio vettoriale la cui dimensione è direttamente proporzionale al numero di prole presenti nel dizionario di termini utilizzato nei documenti: all'interno di questo spazio vettoriale, quindi, i documenti sono rappresentati da vettori di valori reali. Per questo, dato un vettore nello spazio vettoriale, ogni sua coordinata definisce il peso che il termine ad essa associato possiede in relazione al documento e all'intero corpus di documenti.

Graficamente un modello VSM è spesso difficile da rappresentare (già per dizionari con $t > 4$ la rappresentazione in $\mathbb{R}^4$ risulta complessa), ma supponendo di avere un dizionario formato da sole 3 parole, sarebbe possibile visualizzare i documenti in tale spazio VSM tramite una collezione di vettori in $\mathbb{R}^3$ *(fig. 2.6)*.

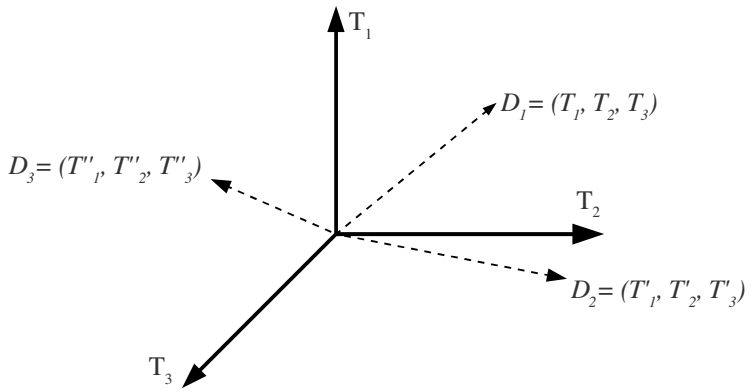


Figura 2.6 *- Rappresentazione di documenti in uno spazio VSM tridimensionale*

2.4.1 Modelli di pesatura

I termini non influiscono tutti allo stesso modo nella caratterizzazione di un documento. Ci sono infatti parole che non sono rilevanti ai fini della comprensione del testo (es. articoli, preposizioni) e che quindi necessitano un trattamento diverso rispetto ad altri più rilevanti.

Possiamo dare ora una definizione più formale del VSM e, più precisamente, del documento rappresentato dal VSM:

ogni vettore d_j del corpus di documenti giace in uno spazio $\mathbb{R}^t$, dove t indica la cardinalità del dizionario impiegato (e, quindi , il numero di termini conosciuti) ed è rappresentato tramite:

$$\vec{d}_j = \sum_{i=1}^{t} x_{ij} \vec{t}_i \tag{2.1}$$

dove t_i è il versore associato all'i-esimo termine ed x_{ij} è il peso associato a tale termine in relazione al documento ed all'intera collezione.

Ovviamente la scelta dei pesi associati ad i vari termini rappresenta un compito fondamentale nel modello VSM, in quanto è proprio essa che determina la rappresentazione finale: essa infatti influisce, tra le altre cose, sulle direzioni associate ai vettori dei documenti nello spazio vettoriale.

Vediamo ora i principali modelli di pesatura.

2.4.1.1 Term Frequency (TF)

Un modo per valutare il peso di un termine *t* all'interno di un documento *d* è quello di considerare la sua *term frequency*: il valore di *t* riferito a *d*, che chiameremo $tf_{t,d}$, è pari al numero di occorrenze di *t* in *d* [3].

In questo modo avremo che termini più frequenti in un documento, otterranno punteggio più elevato.

Lo svantaggio di questo tipo di misura sarà che termini troppo frequenti, che non sono rilevanti ai fini della caratterizzazione del documento otterranno punteggi più elevati di altri più rilevanti.

Una soluzione per attenuare l'effetto del $tf_{t,d}$ è quella di introdurre un altro tipo di misura detta *Inverse Document Frequency*.

2.4.1.2 Inverse Document Frequency (IDF)

Per definire l'idf_t (*inverse document frequency* di *t*), è necessario delineare il concetto di *document frequency* (df_t): la df_t del termine *t* è il numero di documenti nella collezione *c* che presentano una o più occorrenze di *t* [3].

Denotati perciò con df_t la *document frequency* di *t* e con *N* il numero totale di documenti nella collezione, la idf_t è definita come:

$$idf_t = \log \frac{N}{df_t} \tag{2.2}$$

Questa misura permette perciò che termini rari abbiano una idf_t elevata, mentre termini molto frequenti ricevano idf_t con valori più bassi *(fig. 2.7)*.

term	df_t	idf_t
car	18,165	1.65
auto	6,723	2.08
insurance	19,241	1.62
best	25,235	1.5

Figura 2.7 *- Esempio di valori di idf in una collezione di 806791 documenti*

Siamo ora in grado di dare una forma analitica al peso x_{ij} della (2.1) associato all'*i*-esimo termine del *j*-esimo documento e definito come:

$$x_{ij} = tf_{ij} \cdot idf_i \tag{2.3}$$

2.4.2 Modelli di similarità o rilevanza

Dopo aver analizzato i modelli di pesatura nel VSM, vediamo ora come è possibile definire dei modelli di similarità che consentono di valutare la prossimità, in termini vettoriali, di due documenti.

2.4.2.1 Inner product (Prodotto interno)

Dati due documenti d_1 e d_2 rappresentati dai vettori $\vec{V}(d_1)$ e $\vec{V}(d_2)$, l'*inner product* (conosciuto anche come *prodotto interno* o *dot product*) di $\vec{V}(d_1)$ e $\vec{V}(d_2)$ è dato da:

$$\vec{V}(d_1) \cdot \vec{V}(d_2) = \sum_{i=1}^{M} \vec{V}_i(d_1)\, \vec{V}_i(d_2) \tag{2.4}$$

con M numero di dimensioni dello spazio vettoriale.

Un possibile valore di similarità è dato perciò dall'*inner product* di due vettori rappresentanti altrettanti documenti:

$$ip(d_1, d_2) = \vec{V}(d_1) \cdot \vec{V}(d_2) \tag{2.5}$$

2.4.2.2 Cosine similarity

Il modello standard per quantificare la similarità tra due documenti d_1 e d_2 è quello di calcolare la *cosine similarity* [6] della loro rappresentazione vettoriale $\vec{V}(d_1)$ e $\vec{V}(d_2)$

$$cs(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|} \tag{2.6}$$

dove il numeratore rappresenta l'*inner product* dei vettori $\vec{V}(d_1)$ e $\vec{V}(d_2)$, mentre il denominatore è il prodotto delle loro lunghezze Euclidee, definite come

$$\sqrt{\sum_{i=1}^{M} \vec{V_i^2}(d)} \tag{2.7}$$

L'effetto del denominatore della (2.5) è perciò quello di normalizzare le lunghezze dei vettori $\vec{V}(d_1)$ e $\vec{V}(d_2)$ a vettori unitari $\vec{v}(d_1) = \vec{V}(d_1)/|\vec{V}(d_1)|$ e $\vec{v}(d_2) = \vec{V}(d_2)/|\vec{V}(d_2)|$. Possiamo perciò riscrivere la (2.5) come

$$cs(d_1, d_2) = \vec{v}(d_1) \cdot \vec{v}(d_2) \tag{2.8}$$

2.5 Il processo di *Information Retrieval*

Prima di procedere oltre, vediamo in che modo la *cosine similarity* può essere utile alla computazione dei risultati. Per fare ciò, dobbiamo definire alcune tecniche fondamentali per l'interpretazione della *query* utente.

Come abbiamo visto in precedenza, nel VSM ogni documento viene rappresentato da un vettore a *t* dimensioni, dove *t* è la cardinalità del dizionario considerato (tutti i termini presenti nella collezione).

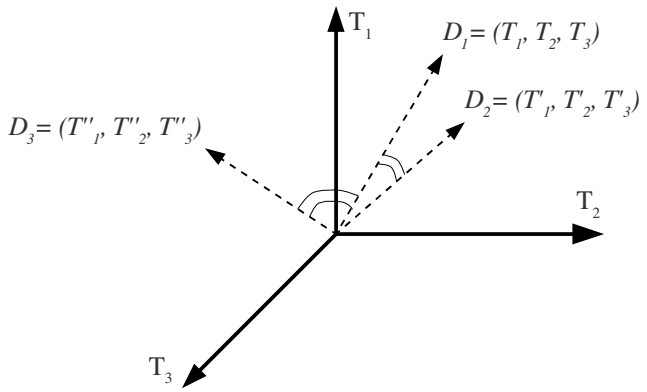


Figura 2.8 *- Similarità tra documenti* [6]

La similarità tra due documenti può essere perciò calcolata sulla base della vicinanza, in termini angolari, dei due vettori che li rappresentano.

Supponiamo di avere la situazione di *figura 2.8* e prendiamo D_1 come documento di riferimento: abbiamo che il documento D_2 è più simile a D_1 piuttosto che D_3, poiché l'angolo tra i vettori D_1 e D_2 è minore di quello tra D_1 e D_3.

Questo concetto di similarità è l'espressione geometrica della forma analitica indicata dalla *cosine similarity*, definita dalla (2.6).

Il salto concettuale da fare è ora quello di considerare la *query* digitata dall'utente come un vettore nello spazio vettoriale generato dalla collezione dei documenti. I termini della *query* prendono così il valore dei termini di un documento, ovvero quello di indicare le dimensioni del vettore corrispondente.

Il processo di *Information Retrieval* si riduce perciò alla computazione della *cosine similarity* tra la *query* e tutti i possibili documenti della collezione: minore sarà l'angolo tra la *query* e un particolare documento, maggiore sarà la rilevanza di quel documento per la *query*.

Ovviamente, soprattutto nei casi di grandi collezioni di documenti, nel processo di *retrieval* verranno filtrati (e restituiti all'utente) solamente i documenti che avranno ottenuto alti punteggi di rilevanza, coerentemente con le scelte effettuate dall'amministratore o, ove possibile, dall'utente.

2.6 Architetture distribuite per l'*Information Retrieval*

Dopo aver analizzato le motivazioni e le tecniche più comuni nell'*Information Retrieval*, passiamo ora a definire l'architettura e le tecnologie utilizzate nella realizzazione di un *IR system*.

Abbiamo affrontato nei capitoli precedenti i passi necessari ad un processo di *retrieval*:

- l'operazione preliminare è quella dell'analisi dei documenti e della valutazione qualitativa dei *field* che li compongono;
- la costruzione dell'indice, tenendo conto di tutti quei task finalizzati ad ottimizzare sia la fase di *indexing*, che quella di *retrieval*;
- la formulazione e l'analisi della *query* (o dell'*information need*);
- la restituzione e lo *scoring* dei risultati.

Nel processo di *retrieval*, è utile suddividere questi task in modo netto, assegnando ogni operazione da effettuare ad un livello architetturale differente, con l'obiettivo di rendere il *search system* più flessibile e portabile.

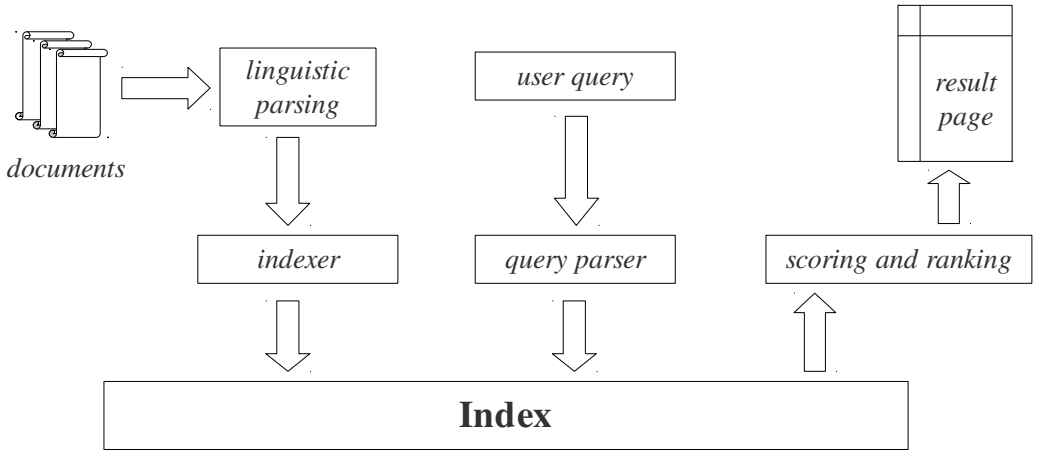


Figura 2.9 *- Un tipico search system* [3]

Come è infatti possibile vedere nella *figura 2.9*, le componenti *user query* e *results page* potrebbero essere gestite da un livello di presentazione, l'*index* da un modulo dedicato ai dati persistenti, mentre *parsing linguistic*, *indexer*, *query*

parser e *scoring and ranking* da un modulo dedicato alla logica funzionale e alla computazione dei dati.

È perciò immediato capire come la *three-tier architecture* possa costituire un'ottima soluzione implementativa sulla quale fondare la nostra applicazione.

2.6.1 Architettura three-tier

L'architettura *three-tier (fig. 2.10)* è una particolare architettura *client-server* nella quale l'interfaccia utente, la logica funzionale e l'accesso ai dati sono sviluppati e gestiti come moduli separati, nella maggior parte delle volte su piattaforme differenti.

Il vantaggio principale nel mantenere separati ed indipendenti i tre livelli architetturali sta nella portabilità e nella riusabilità dei vari componenti. Ad esempio, sarà possibile sostituire il livello di *presentation* (per un cambiamento di tecnologia, per un upgrade dell'applicazione...) senza alterare le funzionalità e l'efficienza del sistema.

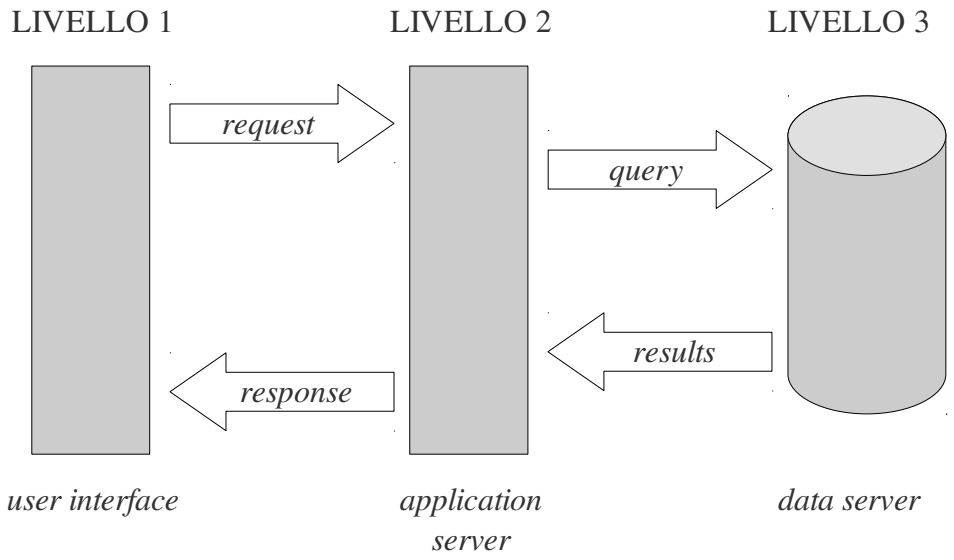


Figura 2.10 *- Architettura three-tier*

L'architettura *three-tier* è composta dai seguenti livelli:

- *presentation tier* (o *front-end*)

- *application tier* (o *business logic*)
- *data tier* (o *back-end*)

Passiamo ora alla presentazione dei tre livelli.

2.6.2 Data Tier

Il *data tier* si occupa della gestione e dell'archiviazione fisica dei dati (ad esempio attraverso un DBMS). Il motivo per cui la gestione dei dati viene resa indipendente dagli altri *tier* sta nel fatto che in questo modo è possibile migliorare scalabilità ed efficienza dell'intero sistema.

2.6.3 Application Tier

L'*application tier* è il livello intermedio di un'architettura *three-tier* e comunica sia con il *presentation tier* che con il *data tier*. Tipicamente viene implementato da un web server (*Jetty* [7], *Tomcat* [8], *GlassFish* [9]) e da tecnologie di elaborazione dati (*Java Servlet* [10], *JavaServer Pages* [11], *CGI* [12], *ASP.NET* [13]). Rappresenta la logica funzionale e l'unità di elaborazione dell'applicazione, computando le richieste fatte dall'utente, interrogando il DBMS e restituendo i risultati al mittente.

2.6.4 Presentation Tier

È il livello che costituisce l'interfaccia utente; infatti questo è l'unico visibile all'*user*. Comunica esclusivamente con l'*application tier.* Nel nostro caso, la scelta è ricaduta sull'utilizzo di un *browser* poiché avevamo la necessità di dover gestire contemporaneamente:

- contenuti statici (pagine HTML)
- contenuti dinamici (elaborati dalla *business logic*)
- contenuti multimediali (filmati)

In più, questa scelta architetturale, permette di localizzare l'intera applicazione su un *server*, eliminando così le operazioni di elaborazione dei dati dalle macchine *client*.

2.7 Tecnologie di base

Nello studio e nell'implementazione di sistemi di IR, un ruolo fondamentale è ricoperto dai *tool* a supporto dell'utente e del programmatore.

Queste tecnologie rappresentano i servizi che permettono di implementare le tre componenti principali di un tipico flusso di IR o di CLIR:

- indicizzazione
- interfaccia utente
- *retrieval* e *ranking*

Nel nostro caso, la scelta di sfruttare la modularità di un'architettura *three-tier*, ha prodotto la necessità di fare uso di diverse tecnologie, facendo si che ognuna di queste si occupasse di una particolare fase del processo di *retrieval*.

Come *presentation layer*, la nostra scelta è stata *Internet Explorer* [14], obbligata dall'utilizzo di *plugin* proprietari non supportati dagli altri *browser*.

La *business logic*, invece, ha visto l'utilizzo di *Apache Tomcat* come web server, mentre tecnologie come *Java Servlet* e *JavaServer Pages* dedicate alle operazioni di elaborazione dei dati.

Infine, tra i vari progetti a disposizione dedicati all'indicizzazione ed il recupero di documenti full-text (*Zettair* [15], *Wumpus Search Engine* [16], *Terrier IR Platform* [17]), abbiamo optato per *Apache Lucene* [18], una potente libreria per *full-text retrieval* della *Apache Software Foundation* [19].

2.7.1 Apache Lucene

Lucene è un progetto *open-source* della *Apache Software Foundation*. Le sue elevate prestazioni e la sua portabilità, l'hanno fatta diventare una delle tecnologie più utilizzate nello sviluppo di sistemi di IR.

I concetti fondamentali in *Lucene* sono *index*, *document*, *field* e *term*.

- Un *index* contiene una sequenza di *documents*;
- un *document* è una sequenza di *fields*;
- un *field* viene identificato da un nome ed è una sequenza di *terms*;
- un *term* è una stringa [20].

Le *query* in *Lucene* viene rappresentata come un particolare *document*. In questo modo è possibile computare la consistenza dei documenti con la *query*

digitata, sulla base della filosofia di similarità definita nel paragrafo 2.5.

La stessa stringa in due *fields* differenti viene considerata un *term* differente. Perciò, i *terms* sono rappresentati da una coppia di stringhe, la prima che si riferisce al *field* di appartenenza, mentre la seconda che rappresenta il testo all'interno del *field*.

L'index di *Lucene* appartiene alla famiglia degli indici conosciuti come *inverted index*; questo perché può tener traccia per un *term* la lista dei *document* che lo contengono.

Lucene prevede inoltre un processo di *tokenization* del testo di un *field*, che può essere evitato per necessità di indicizzare un testo in modo letterale. Nella fase di indicizzazione è possibile impostare dei livelli di *boost* per il *document*, per il singolo *field* e per la *query* nella fase di ricerca.

Il *boost* è un valore assegnato, in fase di *indexing*, ad un particolare *field* o *document* dall'amministratore del sistema, che serve ad evidenziare un *field* piuttosto che un altro. Ad esempio, se in un libro, indicizzato come un documento, si volesse dare più importanza ai termini del titolo, piuttosto che al corpo del libro, basterebbe impostare un livello di *boost* alto per quel *field*. Questo permette a *Lucene* di computare lo *scoring* dei risultati tenendo conto del desiderio di pesatura dell'amministratore.

Per quanto riguarda lo *scoring*, *Lucene* si basa su una combinazione tra il VSM e il *Boolean Model* [21] per determinare la rilevanza di un documento per la *query* dell'utente ed, a livello computazionale, assume la forma:

$$score(q,d) = c(q,d) \cdot qNorm(q) \cdot \sum_{\forall t \in q} (tf_{t,d} \cdot idf_t^2 \cdot b(t) \cdot n(t,d)) \tag{2.9}$$

dove [21]:

- $c(q,d)$ è un valore basato su quanti *terms* della *query q* sono stati trovati nel *document d*;
- $b(t)$ è il valore di *boost* assegnato al *term t* nel momento di formulazione della *query*;
- $qNorm(q)$ è un fattore di normalizzazione della *query* basato sulla formula:

$$qNorm(q) = \left(\sqrt{boost(q)^2 \cdot \sum_{\forall t \in q} (idf_t \cdot boost(t))^2} \right)^{-1} \tag{2.10}$$

che permette di rendere comparabili *scores* tra *query* [21];

- $tf_{t,d}$ e idf_t rispettivamente *term frequency* e *inverse document frequency* definite nei paragrafi 2.4.1.1 e 2.4.1.2;
- $n(t,d)$ incapsula valori di *boost* definiti nel momento dell'*indexing* (*document* e *field*) e fattori di lunghezza dei *field f* nel modo seguente:

$$n(t,d) = boost(d) \cdot lenghtNorm(f) \prod_{\forall f \in d \mid f = t} boost(f) \tag{2.11}$$

2.7.2 Apache Tomcat Web Server

È il *web server* scelto per rappresentare il *presentation layer*. *Tomcat* è un server open source della *Apache Software Foundation* ed implementa le tecnologie *Java Servlet* e *JavaServer Pages* di *Sun Microsystem* [22].

È uno dei *web server* più utilizzati nello sviluppo delle *web application*.

2.7.3 Java Servlet

Le Servlet sono particolari classi Java attraverso le quali è possibile effettuare richieste con il protocollo HTTP. In questo modo è possibile aggiungere contenuti dinamici ad un web server per generare automaticamente pagine HTML (o XML).

Le Servlet rappresentano l'alternativa Java a tecnologie non-Java come CGI e ASP.NET. Il vantaggio nell'utilizzo delle Servlet sta nelle migliori performance (gestione di ogni richiesta come singolo processo) e nella facilità d'uso (possibilità di essere incluse in un *package WAR* come una Web Application).

2.8 Il problema del CLIR

Il rapido sviluppo delle tecnologie di comunicazione, come il World Wide Web, e delle tecniche di IR, hanno permesso a gran parte della popolazione di poter accedere ad informazione precedentemente irraggiungibile. Con questi progressi, è diventato impellente il bisogno di accesso all'informazione in varie lingue, che non siano la propria.

Il CLIR (*Cross-Lingual Information Retrieval*) nasce proprio dall'esigenza di poter rendere accessibile l'informazione agli utenti nella propria lingua e, più in generale, in molteplici lingue.

Facciamo un esempio. Supponiamo che l'utente debba cercare in una

collezione, in cui i documenti sono espressi in linguaggi differenti dalla propria e che non sia in grado in esprimere la *query* in nessuno di questi linguaggi. In questo modo, tutti i termini digitati dall'utente non potranno essere trovati nei documenti, poiché non appartenenti ai dizionari della collezione (es. la parola *"guerra"* non la troviamo nei vocabolari inglesi, tedeschi, francesi...): la ricerca restituirebbe perciò un *empty set*. L'apporto che da un *CLIR system* è quello di mappare la *query* dell'utente, dal *source language* (la lingua di espressione della *query*) ai *target languages* (tutte le possibili lingue della collezione), per poter sollevare l'utente dall'onere della traduzione.

Inoltre, nella grande maggioranza dei casi, queste tecnologie implementano anche una tipizzazione ed una contestualizzazione della *query*, in modo da interpretare al meglio l'*information need* dell'utente.

Oard e Dorr [2] danno varie motivazioni alla ricerca nel CLIR:

- per collezioni che contengono documenti in molte lingue, dove l'espressione della *query* in una lingua sarebbe estremamente inefficiente;
- per documenti che contengono testi in più di una lingua;
- per utenti non in grado di formulare *query* in un'altra lingua differente dalla loro, ma capaci di utilizzare i documenti restituiti.

Per tutte queste ragioni, quella del CLIR è diventata un'area importante nel mondo dell'IR.

2.8.1 La espansione della query come soluzione del CLIR

Supponiamo ora che la *query* sia data in una lingua (es. Inglese) e i documenti indicizzati siano in un'altra lingua (es. Italiano). Un esempio potrebbe essere la *query "Iraqi war"* che, in preparazione al processo di *retrieval*, dovrebbe essere tradotta con *"Guerra in Iraq"*.

Questo task di CLIR viene di solito affrontato con due differenti approcci: statistico o basato sulla conoscenza.

L'approccio statistico è basato su grandi collezioni di *aligned texts* (cioè testi in entrambi i linguaggi, ottenuti attraverso la traduzione diretta della frase). Questa tecnica mostra ottimi risultati in termini di *precision* e *recall*, ma ha il difetto di essere troppo dipendente dalla disponibilità o meno degli *aligned texts*.

L'approccio basato sulla conoscenza o sui dizionari, invece, ha valori minori di *precision* e *recall*, ma non ha bisogno di *aligned texts*. Esso fa uso esclusivamente

di dizionari bilingue e, come l'approccio statistico, soffre di scarsa portabilità e adattamento alle variazioni di dominio.

La soluzione adottata per affrontare le problematiche del CLIR è quella di espandere la *query* utente.

Vediamo come si sviluppa questo processo.

Supponiamo di avere una *query* del tipo *"Berlusconi riferisce in parlamento sulla guerra in Iraq"*. La prima operazione che viene effettuata è quella di identificazione, con l'aiuto di una componente di categorizzazione, di tutti i termini semantici della *query*. In questo modo, si va alla ricerca di termini come *"Berlusconi"* e *"Iraq"*, per potergli assegnare le categorie semantiche corrispondenti (rispettivamente *PersonType* e *PlaceType*). A questo punto, servendosi di una base di sinonimi, è possibile esplodere ogni singolo termine giudicato rilevante, in una lista di termini equivalenti o simili. Questa operazione viene effettuata per individuare il contesto della *query* e, quindi, per scegliere successivamente le traduzioni migliori per un dato termine.

Ad esempio, il termine *"guerra"* potrebbe essere esploso in termini come *"guerra, battaglia, combattimento, conflitto, scontro"*. Ognuno di questi, poi, verrà tradotto con i cosiddetti *termini target* (ovvero, le traduzioni) *"war, battle, engagement, conflict, fight"* dai quali ne verrà selezionato uno, più consistente con il contesto della *query*.

In questo modo, oltre a creare un *mapping* completo della *query* da *source language* a *target language*, abbiamo ottenuto una contestualizzazione più precisa dell'*information need*, da poter sfruttare in fase di *retrieval* puntando a particolari *field* dei documenti.

L'espansione della *query* passa così attraverso task che mirano a risolvere i problemi di disambiguazione del dominio e della scelta dei *termini target* appropriati:

- *semantic disambiguation*: comprensione del dominio attraverso l'analisi del senso delle parole;
- *LSA-based domain modelling*: espansione delle parole in termini che rappresentano lo stesso *topic*;
- *dynamic domain-driven query translation*: traduzione della *query* guidata dal *topic* del contesto.

In questo modo, l'architettura di un *IR system* di *figura 2.9* viene completata con l'introduzione di ulteriori componenti, dedicate al *processing* dei documenti, in fase di indicizzazione, e della *query*, in fase di *retrieval (fig. 2.11)*.

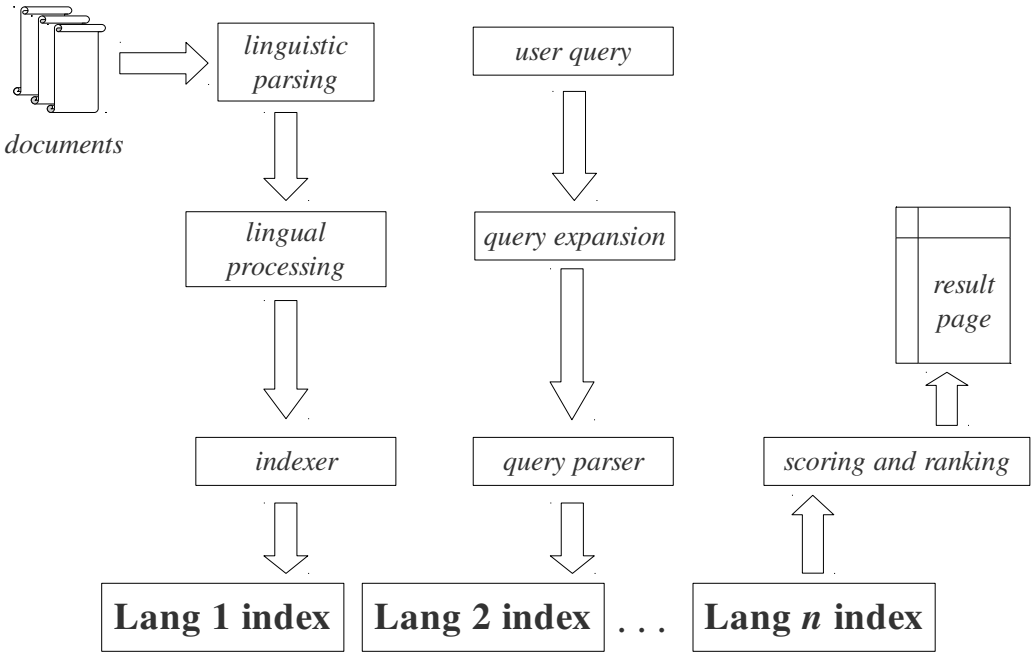


Figura 2.11 *- Un tipico search system*

Approfondiremo il discorso del CLIR più avanti, nel paragrafo 3.3.3, analizzando la sua architettura e le principali tecniche utilizzate.

3 Il progetto di un *front-end* di IR a dati multimediali

Dopo aver analizzato le tecniche di IR e le tecnologie coinvolte, passiamo ora alla presentazione di quello che è stato il lavoro principale della mia tesi.

La produzione di un sistema di IR passa attraverso fasi di sviluppo necessarie alla realizzazione del progetto:

- la comprensione del problema;
- l'analisi dei requisiti;
- la creazione di un modello standard di classificazione dei dati;
- la scelta delle tecnologie;
- la stesura del codice;
- ed, infine, una prima fase di test.

Un *front-end* è un'interfaccia, o applicativo, a servizio dell'utente, che gli permette di effettuare determinate operazioni. Con *front-end* di solito ci si riferisce al *presentation tier* di una web application con architettura *three-tier*; nella mia implementazione, invece, sono stati inclusi anche i livelli di *application* e *data tier*, rendendo così il *front-end*, una web application completa.

3.1 Analisi dei requisiti

Prima di iniziare l'implementazione, occorre capire bene quello che l'applicazione che andiamo a sviluppare deve fare e che servizi deve rendere disponibili.

Nel mio caso, il *front-end* deve poter indicizzare una data classe di contenuti multimediali (e possibilmente essere flessibile al cambiamento della tipologia dei dati) ed effettuare le ricerche sulla collezione dei documenti creata. Inoltre, il sistema deve poter collegare ogni documento alla sua sorgente di contenuti e recuperare l'oggetto dal server per la presentazione all'utente. Le ricerche devono supportare ambienti *cross-lingual* (nel mio caso *inglese* → *italiano*) attraverso la traduzione simultanea della *query* e l'analisi semantica dei suoi termini.

3.1.1 Il repository ANTS

La nostra classe di contenuti è quello che noi chiamiamo il *repository ANTS* ed è la struttura dalla quale estrapoleremo i *metadati* e le informazioni necessarie.

Ogni *repository* rappresenta un contenuto diverso che, nel nostro caso, si riferisce ad un telegiornale della RAI. La visualizzazione dell'interfaccia inclusa nel *repository* è possibile solo con l'utilizzo di un *browser*.

Le problematiche principali nella scansione e analisi dei *repository ANTS* sono dovute proprio alla dislocazione dei *metadati* nei vari file HTML e al fatto che il recupero delle informazioni può essere effettuato solamente grazie a librerie (nel nostro caso *jTidy*) che permettono di visitare una struttura a tag come quella di una pagina HTML.

I dati contenuti nel *repository*, che saranno di nostro interesse, sono principalmente di tre tipi:

- *metadati*: rappresentano le informazioni basilari sul telegiornale (edizione, data di trasmissione, titolo del programma...);
- dati semantici: rappresentano tutti i termini su cui è stata possibile effettuare un'analisi semantica e di categorizzazione (Berlusconi → PersonType, Italia → PlaceType...);
- trascrizioni: rappresentano le trascrizioni del parlato del telegiornale.

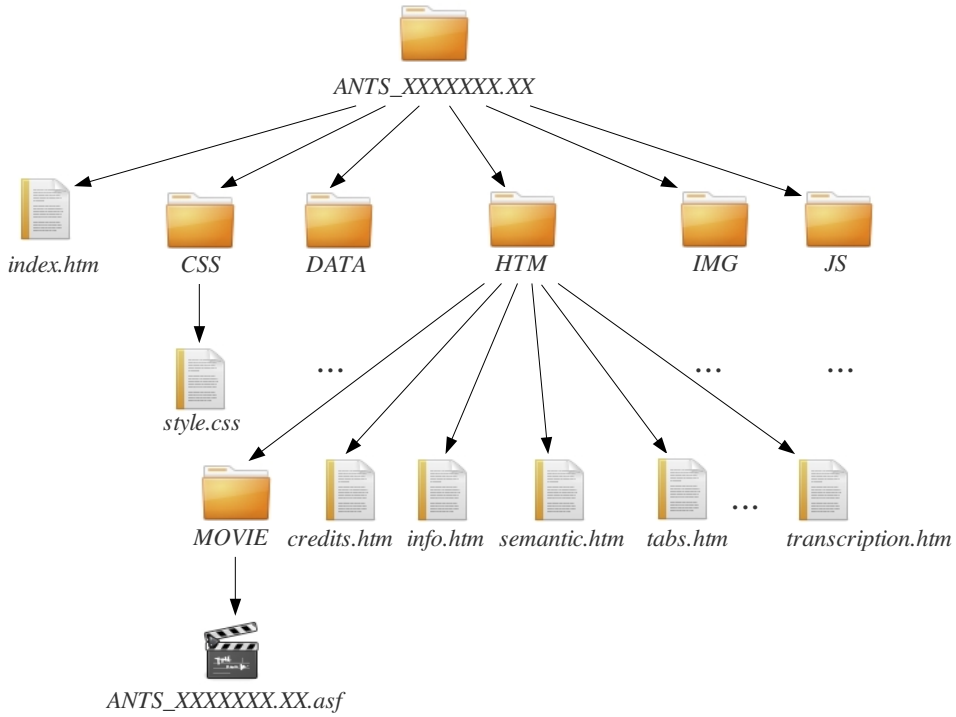


Figura 3.1 *- Struttura di un repository ANTS*

La *figura 3.1* mostra la struttura di un telegiornale ANTS. La cartella principale (*ANTS_XXXXXXX.XX*) viene indicata con l'ID univoco con il quale il telegiornale viene identificato. Scendendo nell'albero del *repository*, notiamo la presenza di una cartella dedicata agli stili *css* (CSS), una cartella per dati vari (DATA), per le immagini di formattazione (IMG) e per gli script JavaScript (JS).

La cartella HTM, invece, contiene i file che ci interessano, in cui possiamo recuperare tutte le informazioni necessarie:

- *MOVIE* → *ANTS_XXXXXXX.XX.asf* è il filmato del telegiornale a cui il *repository* si riferisce;
- *info.htm* ci servirà per ottenere i *metadati* di più basso livello (canale, edizione, orario di trasmissione...);
- *semantic.htm* contiene le informazioni semantiche relative al singolo frammento di filmato (nel nostro caso una *news*);
- in *transcription.htm (fig. 3.2)* possiamo recuperare le trascrizioni del

telegiornale scritte mediante un processo *ASR* (*Automatic Speech Recognition*).

```
<DIV ID="4" NAME="4">
<TABLE><TR>
<TD ID="sec337"></TD>
<TD> di bankitalia ha parlato anche <LABEL
TITLE="persona" name="persona">romano prodi</LABEL> alla
festa dei verdi di <LABEL TITLE="citta_italiana"
name="citta_italiana">bologna</LABEL> la vicenda della
banca d' <LABEL TITLE="paese" name="paese">italia</LABEL>
ha detto il leader del centrosinistra ci sta danneggiando
moltissimo con immagini e siccome l' immagine un fatto
importantissimo bisogna risolvere in <LABEL
TITLE="famiglia" name="famiglia">fretta</LABEL> questa
dice a <LABEL TITLE="famiglia"
name="famiglia">berlusconi</LABEL> ha risposto subito il
presidente della banca centrale europea seguiamo il caso
<LABEL TITLE="famiglia" name="famiglia">fazio</LABEL> ha
detto ma la responsabilita' spetta al <LABEL TITLE="gov"
name="gov">governo</LABEL> e al <LABEL TITLE="gov"
name="gov">parlamento</LABEL> italiano una posizione tra
l' altro e' stata condivisa </TD>
</TR></TABLE>
</DIV>
```

Figura 3.2 *- Frammento di codice HTML di transcription.htm corrispondente ad una news*

Analizziamo ora i file *info.htm*, *semantic.htm* e *transcription.htm* per capire quale informazioni estrarre.

- *info.htm*

  <TD>Channel**</TD><TD>***RAI3***</TD>**
  È il canale in cui è stato trasmesso il telegiornale.
  <TD>Title**</TD><TD>***TG3***</TD>**
  È il titolo del telegiornale (o del programma televisivo).
  <TD>Edition**</TD><TD>***20:00***</TD>**
  È l'edizione del telegiornale.
  <TD>Start Time</TD><TD>2005-09-09/17:00:05:45+120</TD>
  È l'orario di inizio della trasmissione.
  <TD>End Time**</TD><TD>***2005-09-09/17:33:06:20+120***</TD>**
  È l'orario di fine della trasmissione.
  <TD>**Archive ID**</TD><TD>**TO**</TD>
  È l'identificativo dell'archivio nel quale si trova il telegiornale.

<TD>Tape ID</TD><TD>*TECA1756+SUPPORT01756*</TD>
È l'identificativo del supporto fisico nel quale si trova il telegiornale.
<TD>Instance ID</TD><TD>*ANTS_1126512673.91*</TD>
È l'identificativo del telegiornale.

- *semantic.htm*

<DIV ID="1">
È l'identificativo della news all'interno del telegiornale.
<TD>Politica, Partiti, Istituzioni e Sindacati</TD>
È la categoria tematica dell'intera news.
<TD CLASS="section"></TD>
È la sezione semantica in cui tutti i termini sottostanti, appartenenti alla news con ID uguale a 1, sono identificati.
<TD ID="famiglia"></TD>
<TD>
<TABLE>
<TR><TD>abate</TD></TR>
<TR><TD>berlusconi</TD></TR>
<TR><TD>corso</TD></TR>
<TR><TD>davide</TD></TR>
<TR><TD>marcia</TD></TR>
</TABLE>
</TD>
</DIV>

- *transcription.htm*

<DIV ID="10" NAME="10">
Anche in questo caso rappresenta l'id della news, che corrisponde con quello del file semantic.htm
<TD ID="sec879"></TD>
Ogni news è suddivisa ulteriormente in frammenti che hanno come id il secondo in cui il frammento occorre
<TD> l' <LABEL TITLE="paese" name="paese">iraq</LABEL> avra' bisogno dei suoi </TD>
I tag LABEL rappresentano le informazioni semantiche che sono già contenute in semantic.htm

```
        <TD ID="sec881"></TD>
        <TD> dati americani per altri due anni lo ha
        detto il presidente iracheno talebani in visita
        negli stretti in ritiro anticipato ha detto il
        presidente potrebbe portare alla vittoria dei
        terroristi nel nostro paese
        creare una grave minaccia per tutto il mondo ed
        ora torniamo in <LABEL TITLE="paese"
        name="paese">italia</LABEL> andiamo al festival
        dell' unita' di <LABEL TITLE="citta_italiana"
        name="citta_italiana">milano</LABEL> dove si e'
        parlato troppo della redistribuzione delle sul
        set tra <LABEL TITLE="regione"
        name="regione">nord</LABEL> e sud del mondo sente
        </TD>
</DIV>
```

3.2 Produzione di un modello standard per la descrizione dei dati multimediali

Un problema nel processo di indicizzazione dei documenti riguarda la struttura attraverso la quale raccogliere le informazioni.

Ogni volta che viene analizzato un nuovo *repository ANTS*, i dati di interesse devono essere strutturati, in modo che la componente di indicizzazione possa recuperarli facilmente. Un buon progetto della struttura permette anche di poter trattare non solo gli oggetti su cui noi stiamo lavorando, ma, semplicemente cambiando il *parser*, è possibile indicizzare, per esempio, articoli di un quotidiano online, file audio, show televisivi... tutti quanti raccolti in un modello standard di classificazione dei dati.

È proprio questo il reale vantaggio di utilizzare una struttura comune nella fase di indicizzazione: quello di rendere l'intero sistema di IR più robusto e flessibile ai cambiamenti di dominio della collezione di documenti.

3.2.1 Gestione dei metadati e dei dati testuali

Nella quantità di informazioni a nostra disposizione contenuta all'interno del *repository* di dati *ANTS*, è bene fare una distinzione tra *metadati* e dati testuali (o *full-text*).

Una prima differenza tra le due tipologie di dato la troviamo a livello concettuale: mentre il *metadato* rappresenta un'informazione che caratterizza il programma preso in considerazione, il dato *full-text* è parte del telegiornale e forma il suo contenuto.

Un'altra distinzione, conseguenza della precedente, può essere fatta sulla base della lunghezza fisica: il *metadato* ha una taglia di una o due stringhe, il *full-text* può essere un campo formato da centinaia di parole.

Questa premessa indica la necessità di dover gestire i dati facendo un'opportuna distinzione tra queste due tipologie di informazione.

La pratica più comune nello sviluppo di sistemi IR è quella di dedicare un DBMS ai *metadati* e di gestire i dati testuali con un *full-text search engine*; in questo modo si ha una divisione netta tra i due tipi di dato e la necessità di dover effettuare *query* diverse, su collezioni diverse. Inoltre i DBMS, nella maggior parte dei casi, utilizzano una filosofia di *Boolean retrieval*, mentre i *full-text search engines* fanno del *ranked retrieval* il loro punto di forza in termini di efficienza.

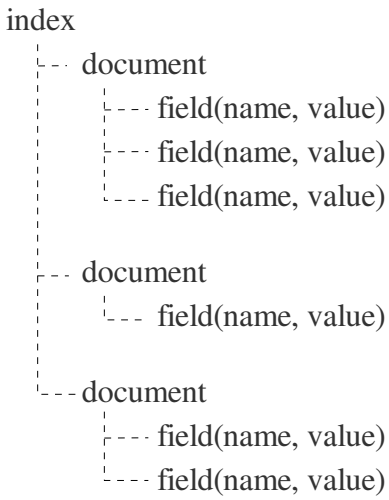
In questa tesi, la scelta è stata quella di non utilizzare DBMS e di includere i *metadati* nei documenti dedicati al *full-text search engine*. Per fare ciò, è stato necessario progettare un modello di dati standard nel quale mappare tutti i *metadati* di un'istanza *ANTS*.

3.2.2 Il modello standard

Prima di presentare il modello pensato per il recupero dei dati dal *repository ANTS*, riprendiamo i concetti di indice e documenti in *Lucene*.

Un indice di *Lucene* è una sequenza di *documents*, mentre i *documents* sono un insieme di *fields*. I *fields* a loro volta sono formati da due stringhe: una per il nome del *field* ed una per il corpus.

Avremo perciò una struttura del tipo:



Il modo più semplice per rappresentare i dati attraverso un modello standard è quello di riprodurre la struttura di *Lucene* con l'aiuto della sintassi XML e mappare i *metadati ANTS* come se fossero *fields* di *Lucene*., seguendo una possibile DTD, come in *figura 3.3*.

```
<!ELEMENT program (doc)>
      <!ELEMENT doc (field+)>
            <!ATTLIST field
                  name CDATA #REQUIRED>
            <!ELEMENT field (#PCDATA)>
```

Figura 3.3 *- DTD di un modello standard di rappresentazione di dati*

Ogni documento, escluso il primo, all'interno dei *tag* **<program>** e **</program>** rappresenta una news o frammento di programma dello stesso telegiornale. Infatti, come si può vedere in *figura 3.6*, il primo documento ha una struttura differente dagli altri due: esso rappresenta *l'header* di un telegiornale e contiene tutti i *metadati* comuni ai documenti successivi *(fig. 3.4)*. La distinzione tra queste due tipologie, viene esplicitata anche dal *field* **<field** name="content"**>** che assume valore **false** nel caso in cui il documento sia l'intestazione e **true** altrimenti.

Il documento *header (fig. 3.4)* riflette tutte le informazioni estratte dal file *info.htm* che rappresentano i *metadati* del programma:

- Instance ID → *programID*
- Channel → *channel*
- Title → *title*
- Edition → *edition*
- Start Time → *sTime*
- End Time→ *eTime*
- Archive ID→ *archID*
- Tape → *tapeID*

```
<program>
      <doc>
            <field name="programID">ANTS_1127979354.08</field>
            <field name="channel">RAI3</field>
            <field name="title">TG3</field>
            <field name="edition">13:30</field>
            <field name="sTime">2005-09-27/12:28:06:82+120</field>
            <field name="eTime">2005-09-27/12:47:52:00+0',</field>
            <field name="archID">TO</field>
            <field name="tapeID">TECA1850+SUPPORTO1850</field>
            <field name="content">false</field>
      </doc>

      ...

</program>
```

Figura 3.4 *- Header di un modello di rappresentazione di dati standard*

Gli altri documenti *(fig. 3.5)*, che, come abbiamo detto prima, rappresentano i frammenti del telegiornale, sono stati pensati con i seguenti *fields*:

- *programID* → È l'identificatore del programma a cui appartiene la news e funge da puntatore all'*header*
- *content* → In questo caso vale **true** poiché il documento rappresenta un contenuto

estratti da *transcription.htm*

- *time* → È il *field* che possiede il riferimento temporale alla news nel telegiornale
- *transcription* → È la trascrizione via ASR del parlato della news

estratti da *semantic.htm*

- *category* → La categoria editoriale semantica della news
- *n fields* che elencano tutti i termini semantici che sono stati trovati nel frammento di testo *transcription*. In questo caso le parole che appartengono alla stessa classe semantica, sono state raggruppate nello stesso *field*, separati da stringhe vuote, mentre le *biwords* (parole il cui senso necessita di due termini) separate da underscore.

```
<program>

        ...

        <doc>
                <field name="programID">ANTS_1127979354.08</field>
                <field name="content">true</field>
                <field name="time">304</field>
                <field name="category">Economia,Credito e Finanza</field>
                <field name="transcription">
                e con l' istat ad agosto salari e stipendi sono aumentati
                del due,nove % rispetto allo scorso anno sono cresciute
                le buste paga di militari e forze dell' ordine
                agricoltori ex pm invece quelle dei dipendenti pubblici e
                degli enti locali della scuola e sanita' rimangono da
                chiudere ancora sottolinea l' istat circa trenta
                contratti nazionali di lavoro e di peso
                </field>
                <field name="gov">istat</field>
                <field name="moneta">peso</field>
        </doc>
        <doc>
                <field name="programID">ANTS_1127979354.08</field>
                <field name="content">true</field>
                <field name="time">505</field>
                <field name="category">Politica,Partiti,Istituzioni e
                Sindacati</field>
                <field name="transcription">
                nessuno vuole vagliare la chiesa replica e la radicale
                capezzone che osserva la situazione italiana e' unica al
                mondo protegge anno col concordato una sola confessione
                religiosa che dispone di straordinari privilegi
                concordata con le sue gerarchie entra a gamba tesa nella
                politica nelle scelte di lutto se si vuole concordate ai
                suoi primi viaggi conclude capezzone lo si rispetti si
                eviti di interferire nell' arena  poli
                </field>
                <field name="citta_italiana">arena poli</field>
                <field name="famiglia">chiesa gamba</field>
        </doc>

        ...

</program>
```

Figura 3.5 *- News di un modello di rappresentazione di dati standard*

```
<program>
        <doc>
                <field name="programID">ANTS_1127979354.08</field>
                <field name="channel">RAI3</field>
                <field name="title">TG3</field>
                <field name="edition">13:30</field>
                <field name="sTime">2005-09-27/12:28:06:82+120</field>
                <field name="eTime">2005-09-27/12:47:52:00+0',</field>
                <field name="archID">TO</field>
                <field name="tapeID">TECA1850+SUPPORTO1850</field>
                <field name="content">false</field>
        </doc>
        <doc>
                <field name="programID">ANTS_1127979354.08</field>
                <field name="content">true</field>
                <field name="time">304</field>
                <field name="category">Economia,Credito e Finanza</field>
                <field name="transcription">
                e con l' istat ad agosto salari e stipendi sono aumentati
                del due,nove % rispetto allo scorso anno sono cresciute
                le buste paga di militari e forze dell' ordine
                agricoltori ex pm invece quelle dei dipendenti pubblici e
                degli enti locali della scuola e sanita' rimangono da
                chiudere ancora sottolinea l' istat circa trenta
                contratti nazionali di lavoro e di peso
                </field>
                <field name="gov">istat</field>
                <field name="moneta">peso</field>
        </doc>
        <doc>
                <field name="programID">ANTS_1127979354.08</field>
                <field name="content">true</field>
                <field name="time">505</field>
                <field name="category">Politica,Partiti,Istituzioni e
                Sindacati</field>
                <field name="transcription">
                nessuno vuole vagliare la chiesa replica e la radicale
                capezzone che osserva la situazione italiana e' unica al
                mondo protegge anno col concordato una sola confessione
                religiosa che dispone di straordinari privilegi
                concordata con le sue gerarchie entra a gamba tesa nella
                politica nelle scelte di lutto se si vuole concordate ai
                suoi primi viaggi conclude capezzone lo si rispetti si
                eviti di interferire nell' arena  poli
                </field>
                <field name="citta_italiana">arena poli</field>
                <field name="famiglia">chiesa gamba</field>
        </doc>
        ...
</program>
```

Figura 3.6 *- Frammento di un modello di rappresentazione di dati standard*

Il *modello standard* di rappresentazione di un telegiornale, mostrato per intero in *figura 3.6*, viene generato automaticamente dai moduli dedicati all'analisi del *repository ANTS*. In questo modo, la componente di indicizzazione potrà importare il modello in formato XML ed analizzare il suo contenuto per estrarre i dati necessari alla costruzione dell'indice di *Lucene*.

3.3 L'architettura del *front-end* ai dati multimediali

La *figura 3.7* mostra l'architettura del *front-end* implementato, evidenziando le singole componenti che sono entrate in gioco nella sua realizzazione.

Come si può vedere, le interfacce utente sono rappresentate da:

- *IndexerGUI* (indicizzazione)
- *IndexSearcherWA* (indicizzazione + recupero)
- *IndexSearcherGUI* (recupero)

Per accedere all'indicizzazione, il *front-end* dovrà per forza usufruire della componente `HTMLParser`, mentre l'accesso all'indice è garantito a tutte le componenti che la richiedano, attraverso l'utilizzo del *package* di *Lucene* `LuceneAPI`.

Nei prossimi paragrafi, analizzeremo con attenzione ogni componente, soffermandoci su quelle le cui operazioni sono di fondamentale importanza.

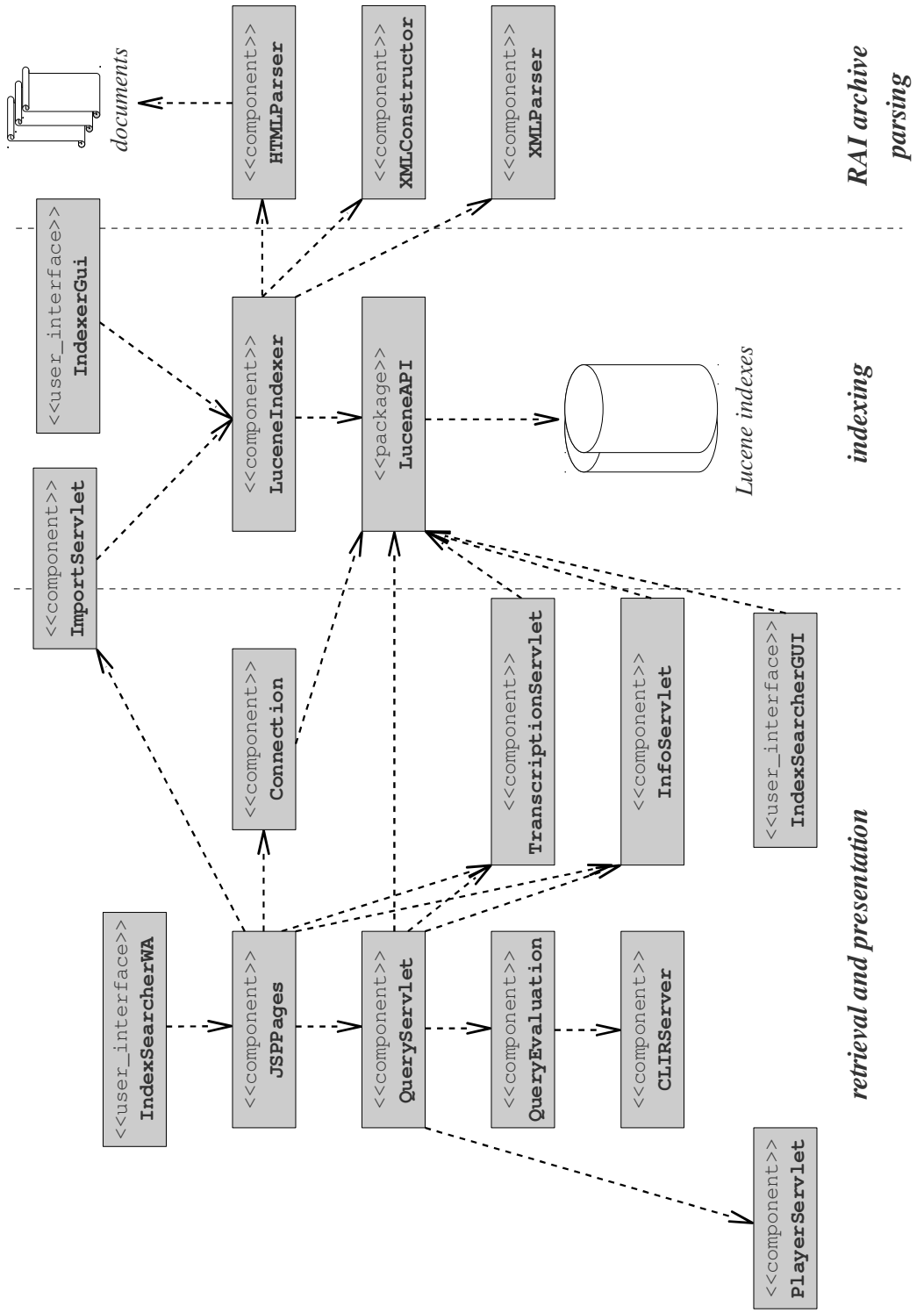


Figura 3.7 *- Architettura del front-end*

Iniziamo l'analisi delle componenti del *front-end*, partendo dalla libreria di *parsers* da me implementata, che risulta fondamentale per la logica operazionale di ogni componente.

L'idea di creare una libreria a parte, dedicata alle operazioni di *parsing*, scaturisce dalla necessità di creare componenti che fossero più portabili e flessibili ai cambiamenti di dominio. Infatti la sostituzione del tipo di dato (*repository RAI*) e del *parser* a lui dedicato, avendo l'accortezza che quest'ultimo generi un modello uguale al nostro, non compromette il successo né della componente di indicizzazione, né di quella di ricerca.

Questa libreria (chiamata semplicemente *Parsers*) è formata da tre classi:

- `HtmlParser`
- `XMLConstructor`
- `XMLParser`

in ordine di utilizzo, ognuna dedicata ad una particolare fase del processo di indicizzazione.

`HtmlParser` *(fig. 3.8)* è la classe che si occupa di analizzare la sintassi del *repository*, utilizzando le librerie *jTidy* per la scansione della struttura HTML. Le maggiori difficoltà nel parsing della struttura ANTS sono state causate dal codice raw, dovuto alla generazione automatica dello stesso. In ogni caso, con opportuni accorgimenti, è stato possibile ovviare il problema e trovare delle valide soluzioni.

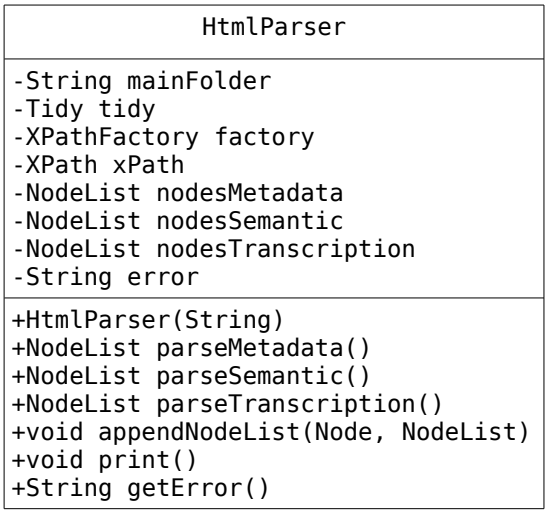


Figura 3.8 *- Descrizione della classe HtmlParser*

Il costruttore inizializza l'oggetto e necessita, come parametro, del path in cui è posizionato il *repository ANTS*. Il parsing vero e proprio avviene con le invocazioni dei metodi `parseMetadata()`, `parseSemantic()` e `parseTranscription()` che si occupano di scorrere le strutture di *info.htm*, *semantic.htm* e *transcription.htm*. I dati vengono poi estratti come `NodeList` ed assegnati agli oggetti `nodesMetadata`, `nodesSemantic` e `nodesTranscription`.

La seconda classe è `XMLConstructor` *(fig. 3.9)*, che si occupa di generare il file XML del modello standard.

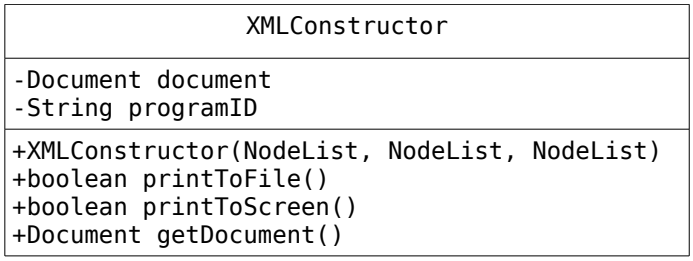


Figura 3.9 *- Descrizione della classe XMLConstructor*

In questo caso il costruttore `XMLConstructor` prende come parametri i tre oggetti `NodeList` creati precedentemente dall'`HtmlParser` e si occupa di costruire, attraverso le librerie *jDom*, il documento XML del modello standard all'interno dell'oggetto `document`. È stato previsto anche un metodo `printToFile()` che consente di scrivere il file XML fisicamente su disco.

La classe `XMLParser` *(fig. 3.10)* interviene al momento del parsing dell'XML generato da `XMLConstructor`, generando un `ArrayList` che riprodurrà tutta la struttura dati del *repository ANTS*.

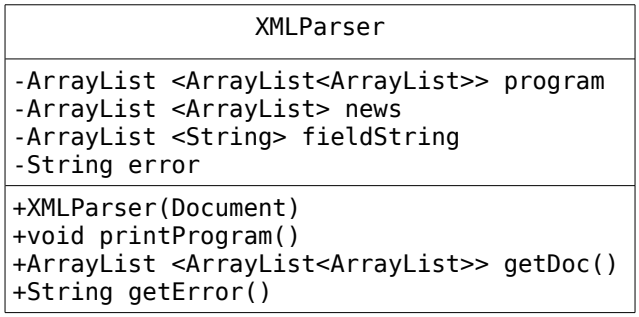


Figura 3.10 *- Descrizione della classe XMLParser*

Le componenti che andrò a presentare completano l'architettura del *front-end* e sono state progettate per essere indipendenti fra loro. Infatti, mentre la componente di indicizzazione si occupa di aggiungere nuovi programmi all'indice, quelle di ricerca offrono un servizio di *retrieval* dell'informazione.

3.3.1 La componente di indicizzazione: INDEXER

L'implementazione della componente di *indexing* è stata effettuata nella prima fase di sviluppo del *front-end*. Il motivo di questa scelta è banale ed emerge facilmente dal più semplice processo di *retrieval*: per poter effettuare un'operazione di recupero dell'informazione, è necessario, come prima cosa, avere (o generare) una collezione sulla quale effettuare le ricerche, soprattutto in un caso come questo, in cui la classe di documenti è stata definita, per forza di cose, con una struttura modellata ad hoc.

Al processo di indicizzazione è stata dedicata una classe, `LuceneIndexer` *(fig. 3.11)*, che, utilizzando i *package* di *Lucene*, aggiunge nuovi documenti all'indice partendo dal modello XML standard.

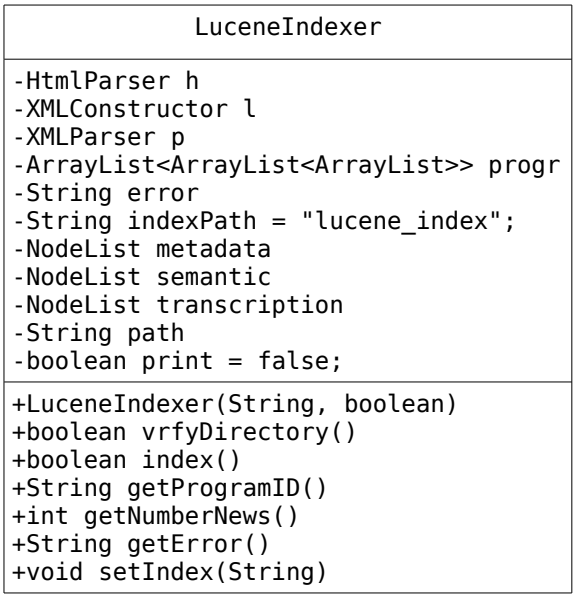


Figura 3.11 *- Descrizione della classe LuceneIndexer*

L'invocazione del costruttore inizializza solamente il path dell'index ed una variabile booleana di controllo (che serve a determinare se stampare o no su file il modello XML). Tutti i compiti dell'indicizzazione sono perciò assegnati ai metodi

`vrfyDirectory()` (verifica l'esistenza e la validità del *repository ANTS*) e `index()` che, invocando le interfacce di *Lucene*, aggiunge i documenti all'indice.

```
for (int i = 0; i < progr.size(); i++) {
       Document doc = new Document();
       for (int j = 0; j < progr.get(i).size(); j++) {
       doc.add(new Field(progr.get(i).get(j).get(0).toString(),
                         progr.get(i).get(j).get(1).toString(),
                         Field.Store.YES,
                         Field.Index.TOKENIZED));
       }
       w.addDocument(doc);
}
w.optimize();
w.commit();
w.close();
```

Figura 3.12 *- Frammento di codice del metodo index()*

Come si può vedere in *figura 3.12*, il telegiornale, che in questo caso è rappresentato da un `ArrayList` (`progr`) generato dalla classe `XMLParser`, viene indicizzato con due cicli `for`: quello più esterno scorre tutti i documenti presenti all'interno del file XML (*header* + news), mentre il `for` interno scorre tutti i *fields*, aggiungendoli al documento corrispondente (`doc.add(new Field(...))`).

I *fields*, a loro volta, vengono creati con l'invocazione del costruttore `Field` che riceve come parametri:

- una `String` che rappresenta il nome del *field*;
- il valore del *field*;
- un parametro che indica se il valore deve essere indicizzato;
- la modalità di indicizzazione del *field*.

Quando il ciclo interno viene consumato, viene aggiunto il documento all'indice tramite l'`IndexWriter w` (`w.addDocument(doc)`). All'uscita dal ciclo esterno, invece, seguono le chiamate che:

- ottimizzano l'indice eliminando eventuali *entries* vuote (`w.optimize()`);
- rendono effettive le modifiche (`w.commit()`);
- terminano la scrittura dell'indice(`w.close()`).

3.3.1.1 IndexerGUI: Java Desktop Application

IndexerGUI è una *Java Desktop Application* che utilizza le librerie grafiche *Java Swing* per l'implementazione della interfaccia utente *(fig. 3.13)*.

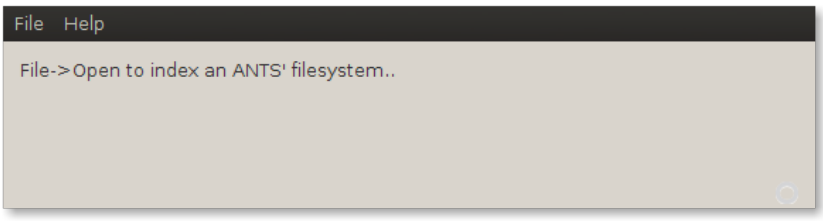


Figura 3.13 *- Schermata di standby di IndexerGUI*

Il suo compito è esclusivamente quello di selezionare il path di un *repository ANTS (fig. 3.14)*, verificare la validità della struttura ed indicizzare il programma.

Queste ultime due operazioni vengono realizzate grazie alle interfacce pubbliche della classe `LuceneIndexer` presentata nel paragrafo precedente.

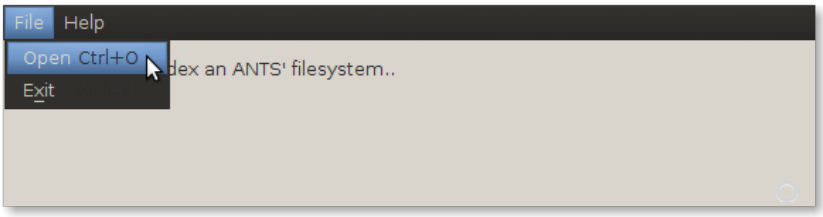


Figura 3.14 *- Modalità di selezione del path*

Il successo dell'indicizzazione verrà stampato a video e, con esso, l'identificatore del telegiornale appena aggiunto e il numero di notizie in esso presenti *(fig. 3.15)*.

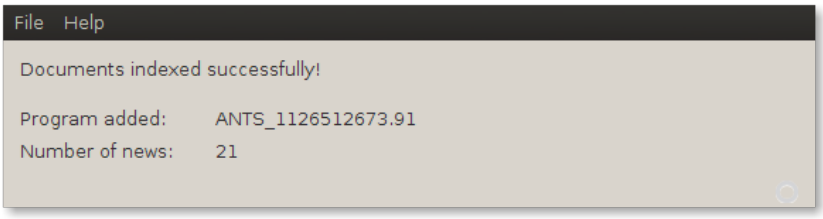


Figura 3.15 *- Indicizzazione avvenuta con successo*

Nel caso in cui i documenti non siano stati aggiunti per l'occorrenza di un problema (es. directory non valida, index corrotto o non trovato...), l'applicazione stamperà un messaggio con l'errore riportato *(fig. 3.16)*.

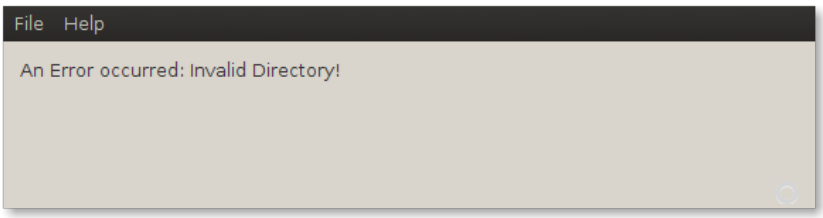


Figura 3.16 *- Indicizzazione fallita*

3.3.2 La componente di ricerca

Il *front-end* si completa con la componente di ricerca: *IndexSearcherGUI* e *IndexSearcherWA*. Queste due applicazioni permettono all'utente di formulare una *query* e di ottenere i risultati ordinati in base al punteggio ricevuto.

La tecnologia utilizzata per il processo di *retrieval* è, in entrambi i casi, *Lucene*, che si occupa di pesare i termini della *query* ed assegnare i punteggi ai risultati.

L'implementazione della componente di ricerca ha prodotto due tool indipendenti che mettono a disposizione dell'utente vari servizi e che, come vedremo nel caso di *IndexSearcherWA*, vanno ad ampliare le funzionalità del recupero ed a sovrapporsi alla componente di indicizzazione.

Questa parte del progetto è la reale motivazione della tesi e l'obiettivo degli studi affrontati precedentemente.

Andiamo ora ad analizzare i singoli tool nello specifico, presentando screenshot e frammenti di codice implementato.

3.3.2.1 IndexSearcherGUI: Java Desktop Application

È la prima delle due applicazioni di *retrieval* implementate. Sebbene sia completa di tutte le funzionalità di recupero, essa è servita esclusivamente per comprendere l'utilizzo delle interfacce di *retrieval* di *Lucene* e per effettuare dei test sulla *query* e sull'indice.

IndexSearcherGUI è un'applicazione Java in cui l'interfaccia è implementata, come in *IndexerGUI*, con le librerie grafiche *Java Swing*.

Dopo aver fornito all'applicazione il path dell'indice (*File* → *Open index*), lo scenario che si presenta all'utente è un pannello di standby in cui è possibile iniziare a digitare *query*, possibilmente selezionando vari parametri *(fig. 3.17)*.

La sintassi della *query* è quella di *Lucene* e permette di specificare il *field* in cui cercare un preciso termine (*nome_field:testo_da cercare*), utilizzare *wildcard queries* (*test**) o esplicitare gli operatori booleani con cui i termini verranno elaborati (*term1 AND term2 OR term3*). Omettendo questa sintassi, sarà comunque possibile selezionare il *field* di default, nel quale *Lucene* andrà a cercare i termini digitati e l'operatore predefinito da utilizzare tra un termine e l'altro della *query*.

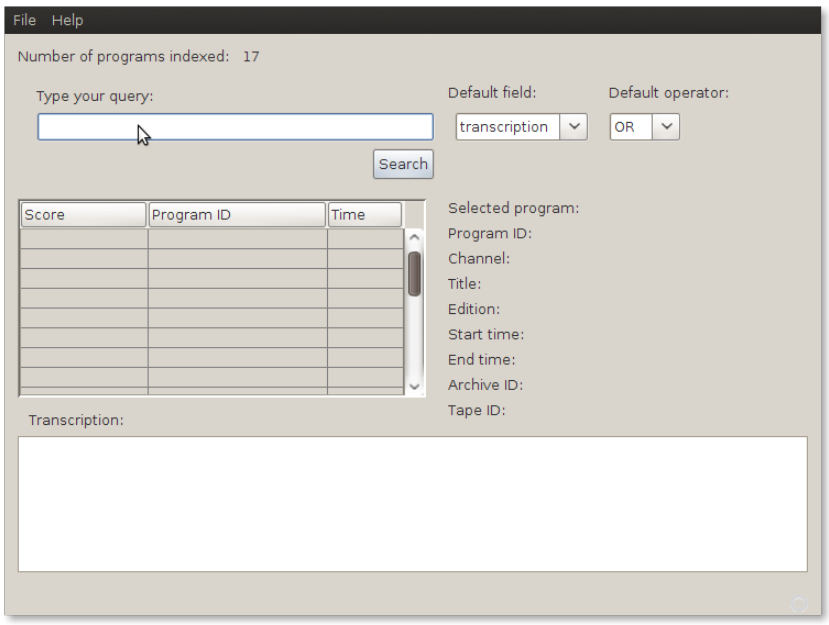


Figura 3.17 *- Pannello di standby dell'applicazione IndexSearcherGUI*

Nel momento di invio della *query*, viene attivata la procedura che, utilizzando le interfacce e gli oggetti di *Lucene*, inizializza il *parser* della *query* e recupera i risultati *(fig. 3.18)*.

```
QueryParser qp = new QueryParser(Version.LUCENE_CURRENT,
               (String) jComboBox1.getSelectedItem(), analyzer);
 if (jComboBox2.getSelectedItem().toString().equals("AND"))
       qp.setDefaultOperator(QueryParser.Operator.AND);
 else
       qp.setDefaultOperator(QueryParser.Operator.OR);
 q = qp.parse(jTextField1.getText());
 collector = TopScoreDocCollector.create(20, true);
 searcher.search(q, collector);
 hits = collector.topDocs().scoreDocs;
 for (int i = 0; i < hits.length; ++i) {
       jTable1.setValueAt(hits[i].score, i, 0);
       jTable1.setValueAt(
                     searcher.doc(hits[i].doc).get("programID"),
                     i, 1);
       jTable1.setValueAt(
                     searcher.doc(hits[i].doc).get("time"), i,
                     2);
}
```

Figura 3.18 *- Processo di retrieval nel codice*

Elaborato il processo di *retrieval*, i risultati verranno stampati in una `jTable` che permetterà di selezionare una entry e visualizzare la trascrizione della news ed i *metadati* del corrispondente documento *header (fig. 3.19)*.

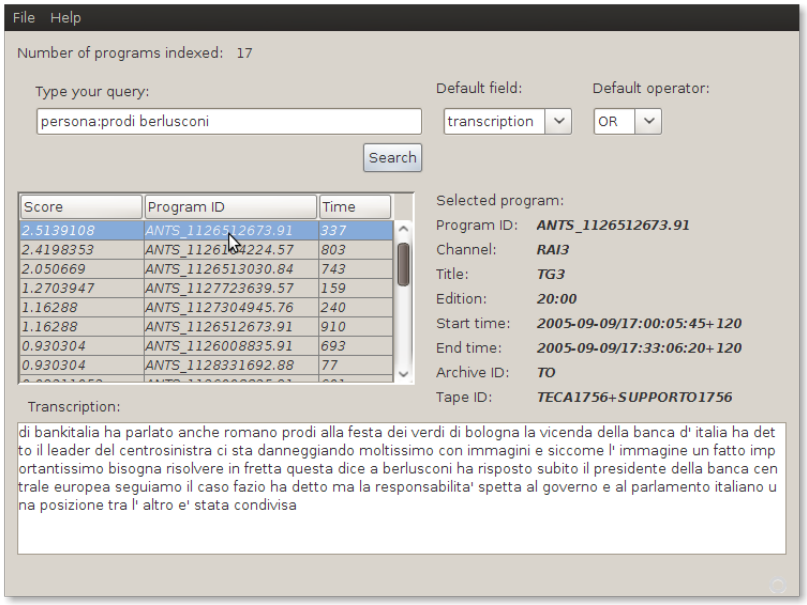


Figura 3.19 *- Visualizzazione dei risultati*

3.3.2.2 IndexSearcherWA: Java Web Application

L'*IndexSearcherWA* è l'applicazione più complessa e quella più completa tra le altre implementate. Infatti, oltre a fornire un servizio di ricerca dell'informazione, permette di visualizzare sul browser il contenuto multimediale a cui un risultato fa riferimento, supporta il *retrieval* multilinguale ed offre anche la possibilità di importare nuovi telegiornali, aggiungendoli all'indice.

Nella sua implementazione mi sono servito di una architettura *three-tier*: questa scelta permette di avere massima portabilità, grazie al paradigma *client-server*. L'applicazione sarà perciò localizzata su un *server*, al quale i *client* si collegheranno ed invieranno le richieste attraverso il protocollo HTTP: saranno perciò eliminate installazioni di software e l'interfaccia utente potrà essere visualizzata all'interno del browser.

Come abbiamo già anticipato alla fine del capitolo 2, la scelta delle tecnologie utilizzate è ricaduta su *Apache Tomcat* come web server, le *Java Servlet* per le operazioni computazionali e *Apache Lucene* per la persistenza e la gestione dei dati.

L'applicazione è perciò formata classi Java e da pagine HTML e JSP (*JavaServer Pages*).

Tra le classi Java troviamo quelle semplici e le *servlet*, che estendono la classe astratta `HttpServlet`.

- `Connection`: è una classe Java pura che serve per stabilire una connessione con l'indice di *Lucene* e con il *CLIR server*, del quale parleremo più avanti.

```
SAXBuilder builder = new SAXBuilder();
Document document = builder.build(new
                              File("conf/index_searcher_conf.xml"));
Element root = document.getRootElement();
Element addressElem = root.getChild("address");
Element hostElem = root.getChild("host");
Element portElem = root.getChild("port");
address = addressElem.getText();
host = hostElem.getText();
port = Integer.parseInt(portElem.getText());
analyzer = new StandardAnalyzer(Version.LUCENE_CURRENT);
index = FSDirectory.open(new File(address));
searcher = new IndexSearcher(index, true);
```

Figura 3.20 *- Costruttore della classe Connection*

  Le impostazioni di connessione sono definite in un file XML (*index_searcher_conf.xml*) che permette di indicare il percorso nel quale è possibile trovare l'indice sulla macchina, l'host e la porta alla quale connettersi per usufruire dei servizi del CLIR. Il frammento di codice in *figura 3.20* mostra il costruttore `Connection()` e le operazioni che compie nell'inizializzare le connessioni.

- `LuceneIndexer`: classe di cui abbiamo già ampiamente parlato nel paragrafo 3.3.1
- `ImportServlet`: è una classe *servlet* che gestisce l'import di nuovi telegiornali nell'indice. Si limita ad inizializzare un oggetto `Connection` (dal quale riceve l'indirizzo dell'indice) ed uno di tipo `LuceneIndexer`, che, come abbiamo visto, si occupa di indicizzare i nuovi documenti. Genera poi la pagina HTML di risposta alle operazioni di *indexing*, con eventuali messaggi di errore.
- `InfoServlet`: viene chiamata una nuova *servlet* di questo tipo, ogni qualvolta sia necessario ricaricare i *metadati* relativi ad una particolare notizia. È stato previsto, infatti, che cliccando su una news, vengano aggiornati i *metadati* contenuti nel file *header* corrispondente e la sua trascrizione. Nell'istante di creazione viene composta una *query* sull'indice

che cerca tutti i documenti con `programID` uguale al riferimento contenuto nella notizia ed il valore `false` nel *field* `content`. I risultati vengono perciò computati e formattati in una pagina HTML che la *servlet* genera in modo dinamico.

- `TranscriptionServlet`: svolge le stesse operazioni della *servlet* precedente, con la differenza che, invece di estrarre i *metadati*, essa si occupa di visualizzare la trascrizione relativa alla notizia selezionata.
- `PlayerServlet`: viene creata nel momento in cui l'utente voglia vedere il contenuto multimediale relativo alla news selezionata. Genera perciò una pagina HTML che contiene il player multimediale, passandogli il percorso nel quale è possibile trovare il filmato e l'offset dal quale farlo partire per puntare direttamente alla news.
- `QueryEvaluation`: è una classe Java semplice che si occupa dell'espansione della *query* (e della sua eventuale traduzione), sfruttando i servizi del CLIR. Vedremo la sua implementazione più avanti, nel paragrafo 3.3.3.
- `QueryServlet`: è la *servlet* che, ricevendo la *query* dall'oggetto `QueryEvaluation`, la passa al *searcher* di *Lucene* e formatta i risultati in una pagina HTML generata dinamicamente.

Le pagine HTML incluse nel progetto non hanno una grande rilevanza poiché prive di codice computazionale. Esse, infatti, sono incaricate esclusivamente di generare l'interfaccia utente. Vedremo invece le JSP, in grado di generare in modo dinamico la sua struttura HTML, grazie al codice Java incapsulato.

La main page *(fig. 3.21)* dell'applicazione è composta da un set di frame, uno per ogni tipo di servizio offerto. Come abbiamo detto, le pagine più interessanti sono le JSP, ovvero quella di ricerca (in alto a destra) e quella in cui viene visualizzato l'indice (in basso a sinistra), mostrando i singoli telegiornali contenuti e la loro struttura.

Nel frame dedicato alla ricerca, abbiamo un *text field*, in cui digitare la *query*, tre *combo box*, per impostare i vari parametri di ricerca, il bottone per avviare il processo di ricerca e quello per intraprendere l'importazione di nuovi telegiornali.

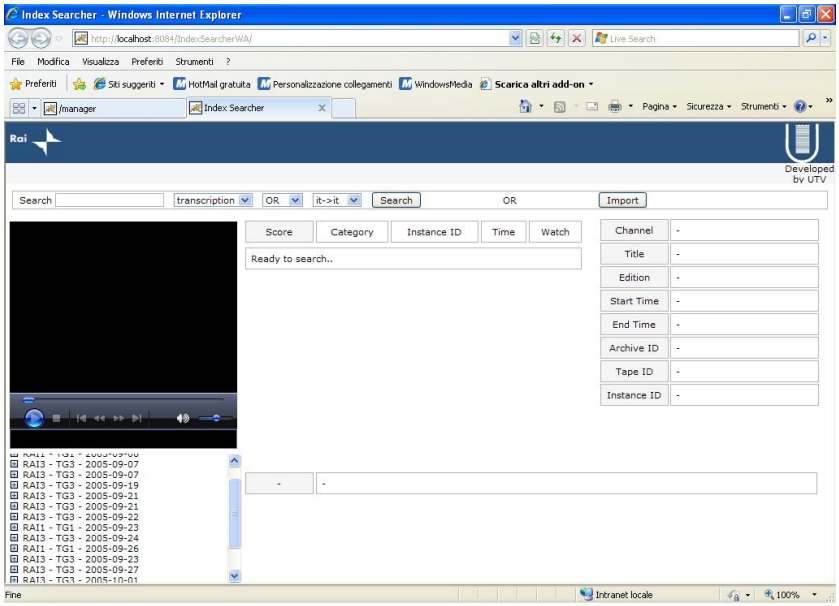


Figura 3.21 *- IndexSearcherWA*

Il frammento di codice in *figura 3.22* mostra il riempimento automatico della prima *combo box*, ovvero quella in cui vengono elencati tutti i *fields* disponibili nell'indice. Il valore selezionato verrà utilizzato da *Lucene* come *field* di default nel processo di ricerca.

```
<SELECT class="button" name="fields" size="1">
      <%for (int i=0; i<c.getFields().size(); i++) {
            if (!c.getFields().get(i).equals("transcription")) {
      %>
      <OPTION class="button" value="
            <%out.print(c.getFields().get(i)); %>">
            <%out.print(c.getFields().get(i)); %>
      </OPTION>
      <%     } else {%>
      <OPTION class="button" value="
            <%out.print(c.getFields().get(i)); %>"
                              selected="selected">
            <%out.print(c.getFields().get(i)); %>
      </OPTION>
      <%}}%>
</SELECT>
```

Figura 3.22 *- Riempimento della prima combo box*

Anche la pagina che mostra l'indice viene generata dinamicamente all'apertura dell'applicazione visualizzando la struttura editoriale del telegiornale in modo automatico. Tutto questo è possibile grazie alle *JavaServer Pages*, che permettono di incapsulare codice Java all'interno di pagine scritte con la sintassi HTML *(fig. 3.23)*.

```
<%for (int i=0; i<c.getCategories().size(); i++) {
       out.println("<TABLE cellpadding='0' cellspacing='0'>" +
       "<TR>" +
       "<TD>" +
       "</TD>" +
       "<TD class='datetxt' onclick=refresh_infos('" +
              c.getCategories().get(i).get(0)+"');>" +
              c.getCategories().get(i).get(1)+" - " +
              c.getCategories().get(i).get(2)+" - " +
              c.getCategories().get(i).get(3) +
       "</TD>" +
       "</TR>" +
       "</TABLE>" +
       "<TABLE cellpadding='0' cellspacing='0'>" +
       "<TBODY id='"+c.getCategories().get(i).get(0)+"'>" +
       "<TR>" +
       "<TD class='globaltxt' height='4'>" +
       "</TD>" +
       "</TR>");
       int count = 0;
       for (int j=4; j<c.getCategories().get(i).size(); j=j+2) {
              out.println("<TR>" +
              "<TD>" +
              "</TD>" +
              "<TD id='sec"+c.getCategories().get(i).get(j+1)+"'>"+
              "</TD>" +
              "<TD>" +
              "<DIV id='"+count+"' class='segment_item'>" +
              "<TABLE cellpadding='0' cellspacing='0'>" +
              "<TR>" +
              "<TD class='menu' onclick=refresh('" +
                     c.getCategories().get(i).get(0)+"','" +
                     c.getCategories().get(i).get(j+1)+"');>"+
                     c.getCategories().get(i).get(j)+"</TD>" +
              "</TR>" +
              "</TABLE>" +
              "</DIV>" +
              "</TD>" +
              "</TR>");
              count++;
       }
       out.println("</TBODY>" +
       "</TABLE>");
}%>
```

Figura 3.23 *- Creazione del menu e della struttura dell'indice*

3.3.3 La componente di Cross-Lingual search

Come abbiamo visto, grazie al *CLIR server* è possibile ottenere un ambiente multilinguale, in cui la traduzione della *query* viene fatta a *run-time*.

In questo modo si ottiene un sistema di IR in cui sia la *query* che i documenti indicizzati possono essere espressi in lingue differenti, togliendo così all'utente l'onere della traduzione.

Tutto questo è possibile grazie all'espansione della *query* che, oltre a tradurre il testo digitato, analizza i termini presenti, assegnando un valore semantico, ove possibile, ad ognuno di essi.

3.3.3.1 CLIR Server (CLIR)

Il *CLIR server* è un *engine* che offre i servizi necessari a garantire un *retrieval* multilinguale dei dati. L'interazione con l'utente è effettuata mediante un meccanismo di *querying* in linguaggio naturale [23]. Il CLIR riceve dall'utente la *query* attraverso il *presentation layer* (interfaccia utente) e serve al motore di *querying* di *Lucene* l'espressione arricchita con le seguenti informazioni:

- Categoria della *query*
- *Named Entities* (ovvero, categorie dei singoli termini, indipendenti dal linguaggio)
- *Ontological Entries* (ovvero, termini conosciuti a priori, anch'essi indipendenti dal linguaggio)
- *Common nouns* (ovvero, nomi comuni, dipendenti dal linguaggio).

Fornisce, inoltre, possibili traduzioni dei *common nouns*, in base ai linguaggi sorgente e obiettivo desiderati [23]. La *query* così espansa, viene restituita all'utente in modo da poter essere modificata con eventuali correzioni o utilizzata per ottenere una lista di risultati che meglio rappresentano l'*information need* dell'utente.

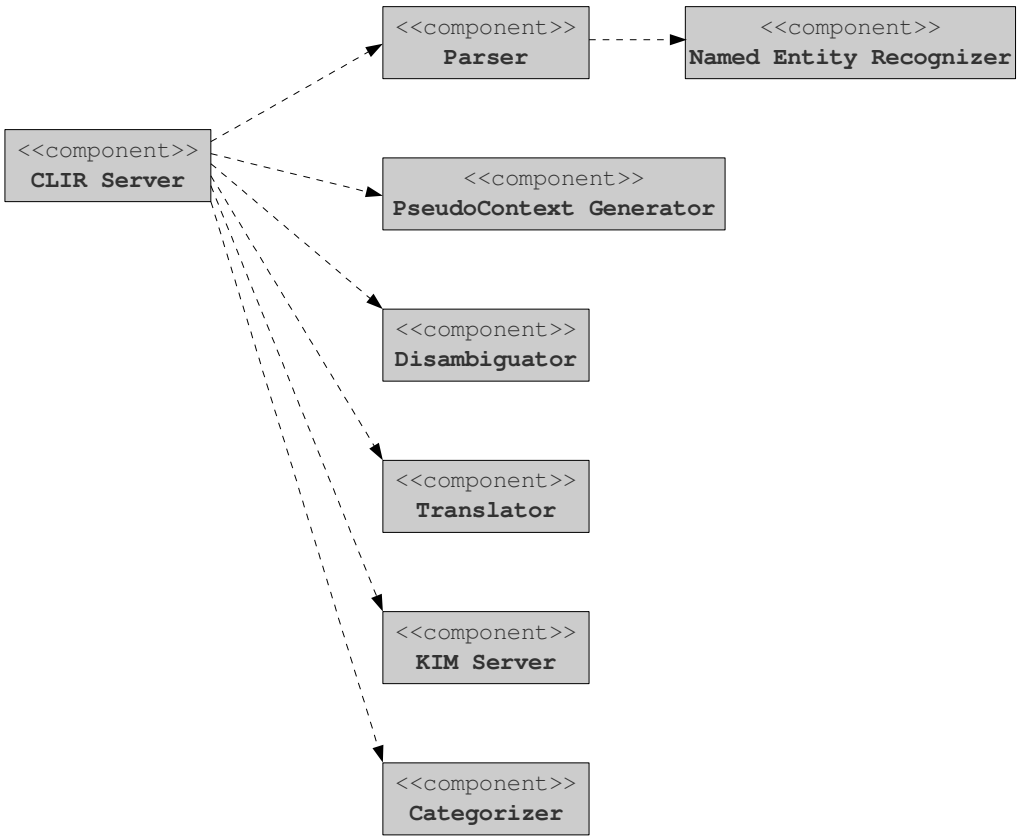


Figura 3.24 *- Architettura del CLIR* [23]

L'architettura del CLIR *(fig. 3.24)* è formata da varie componenti, ognuna dedicata ad un particolare servizio offerto all'utente:

- `Parser`: estrae *Named Entities* e *Common Nouns* dalla *query*
- `Pseudo Context Generator`: rileva per un termine target *t*, i termini più rilevanti che occorrono insieme a *t*
- `Disambiguator`: espande i termini nel linguaggio sorgente per eliminare ambiguità semantiche
- `Translator`: traduce i termini dalla lingua sorgente a quella *target*
- `KIM Server`: individua i termini ontologici nella *query*
- `Categorizer`: assegna una categoria alla *query*

Il CLIR comunica attraverso una porta configurabile (nel nostro caso la 5678) ed inviando e ricevendo informazioni mediante uno *stream XML*. Il *CLIR client* formatta la *query* inserita dall'utente come uno *stream XML* in cui viene tenuto traccia anche delle lingue *source* e *target* e riceve dal *CLIR server* sempre uno *stream XML* con i dati di espansione della *query* [23].

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<query_session id="1">
   <request>
      <gui_state>
         <user_query>guerra sanguinosa in Irak</user_query>
         <gui_language>it</gui_language>
         <target_language>en</target_language>
         <interaction_level>0.5</interaction_level>
      </gui_state>
   </request>
   <response>
      <query_category confidence="0.0">NO CATEGORY</query_category>
      <query_evaluation confidence="1.0" return_state="OK">
         NO MEX
      </query_evaluation>
      <named_entities/>
      <ontology_entries/>
      <nouns/>
   </response>
</query_session>
```

Figura 3.25 *- Input XML Stream* [23]

La *figura 3.25* mostra un esempio di *stream* inviato dal *CLIR client* al *CLIR server*. Nel tag `<user_query>` c'è la *query* inserita dall'utente, mentre `<gui_language>` e `<target_language>` fanno riferimento rispettivamente al linguaggio *source* e a quello *target*. Il *CLIR server* estrae le informazioni da questo *stream* e fornisce le traduzioni in base al valore inserito nel tag `<interaction_level>`, che controlla la precisione nel processo di espansione dei singoli termini [23].

La *figura 3.26*, invece, mostra la riposta del *CLIR server* alla richiesta del *client*. In questo *stream XML* vengono inserite le seguenti informazioni:

- la categoria della *query*, inserita nel tag `<query_category>` con il relativo livello di confidenza `confidence`
- la lista delle *Named Entities*, inserite nel tag `<named_entities>`
- per ogni *NE*, viene fornito il termine con la relativa categoria semantica

- i termini ontologici trovati, inseriti nel tag `<ontology_entries>`
- la traduzione proposta per ogni termine comune, tra i tag `<nouns>`
- per ogni traduzione viene fornita una lista di sinonimi, il termine originale e le traduzioni proposte

In *figura 3.26* è possibile vedere come il livello di confidenza espresso dall'attributo `confidence` del tag **`<query_evaluation>`** influisca sullo stato delle risposte al *client*. Infatti, avendo impostato un *threshold* di 0.9, ogni qualvolta questo livello di confidenza non verrà raggiunto, il *server* restituirà un *warning*, che in questo caso avvisa sulla non attendibilità della traduzione (`UNRELIABLE_TRANSLATION`).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<query_session id="1">
   <request>
      <gui_state>
         <user_query>guerra sanguinosa in Iraq</user_query>
         <gui_language>it</gui_language>
         <target_language>en</target_language>
         <interaction_level>0.5</interaction_level>
      </gui_state>
   </request>
   <response>
      <query_category confidence="1.0">
         urn:x-prestospace-mad:cs:GenreCS:2005:FOR
      </query_category>
      <query_evaluation confidence="0.8595577"
         return_state="UNRELIABLE_TRANSLATION">
         Warning: This question was too complex to be properly
         translated. If you need translations, please try to revise
         your question adding more information, ie. new words, or
         using different (that is less ambiguous ) words.
      </query_evaluation>
      <named_entities>
         <ne surface="Iraq" type="mp7:PlaceType"/>
      </named_entities>
      <ontology_entries/>
      <nouns>
         <noun surface="guerra" lemma="guerra">
            <translations best="1167074" confidence="0.42132694">
               <sense id="1167074" gui_language_lemmas="guerra"
                  target_language_lemmas="war,warfare"/>
               <sense id="1102789" gui_language_lemmas="guerra"
                  target_language_lemmas="strife"/>
               <sense id="896789" gui_language_lemmas="battaglia,
                        combattimento,conflitto,guerra,lotta,scontro"
                  target_language_lemmas="battle,
                        conflict,fight,engagement"/>
               <sense id="13183809"
                  gui_language_lemmas="discordia,
                        disunione,guerra,zizzania"
                  target_language_lemmas="discord,strife"/>
            </translations>
         </noun>
         <noun surface="sanguinosa" lemma="sanguinosa"/>
         <noun surface="Iraq" lemma="iraq">
             <translations best="8785730" confidence="1.0">
                <sense id="8785730" gui_language_lemmas="iraq"
                target_language_lemmas="Iraq,Republic_of_Iraq,Al-
                        Iraq,Irak"/>
             </translations>
         </noun>
      </nouns>
   </response>
</query_session>
```

Figura 3.26 *- Output XML Stream* [23]

3.3.3.2 Integrazione del CLIR

Per poter utilizzare i servizi del *CLIR server* è stato necessario implementare una classe che potesse gestire le richieste e le risposte del server. In particolare, l'integrazione del servizio è stata resa possibile grazie alla classe `Connection`, che rende disponibili i parametri di connessione al server, e alla classe `QueryEvaluation`. Quest'ultima formatta la *query* dell'utente come lo *stream XML* di input che abbiamo visto nel paragrafo precedente e parsa l'output per l'espansione della *query*.

Ma andiamo con ordine.

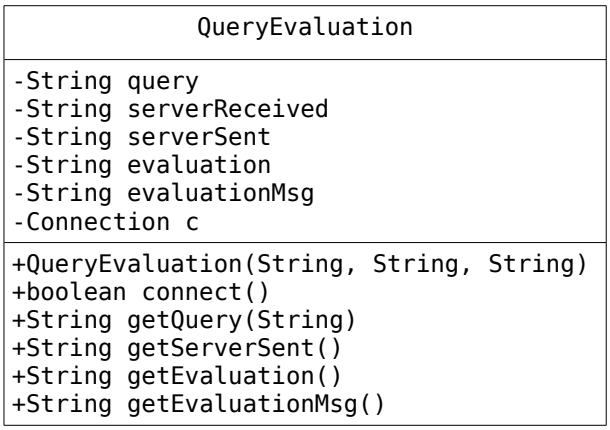


Figura 3.27 *- Descrizione della classe QueryEvaluation*

La *figura 3.27* mostra la struttura della classe `QueryEvaluation`. I metodi più importanti, quelli che svolgeranno il lavoro maggiore, sono:

- il costruttore `QueryEvaluation`
- il metodo `connect()`
- `getQuery(String)`

Il costruttore della classe `QueryEvaluation` *(fig. 3.28)* inserisce la *query* e le lingue (sorgente e target) all'interno di uno scheletro XML che riproduce la struttura dello *stream* di input. L'oggetto di tipo `Connection`, invece, ci servirà per impostare l'host e la porta del server.

```
public QueryEvaluation (String q, String guiLanguage, String
targetLanguage) {
   query = q;
   serverReceived = "<?xml version='1.0' encoding='ISO-8859-1'?
       ><query_session
       id='1'><request><gui_state><user_query>"+
       q+"</user_query><gui_language>"+
       guiLanguage+"</gui_language><target_language>"+
       targetLanguage+"</target_language><interaction_level>"+
       "0.5</interaction_level></gui_state></request>"+
       "<response><query_category confidence='0.0'>NO CATEGORY"+
       "</query_category><query_evaluation confidence='1.0'"+
       "return_state='OK'>NO MEX"+
       "</query_evaluation><named_entities/>"+
       "<ontology_entries/><nouns/></response></query_session>";
   c = new Connection();
}
```

Figura 3.28 *- Costruttore di QueryEvaluation*

Il metodo `connect()` *(fig. 3.29)* effettua la connessione al server (`Socket sock = new Socket(c.getHost(), c.getPort())`), invia lo *stream input* (`writer.write(serverReceived)`) e salva nella variabile `serverSent` lo *stream output* generato dal *CLIR server*.

```
Socket sock = new Socket(c.getHost(), c.getPort());
OutputStream out = sock.getOutputStream();
InputStream in = sock.getInputStream();
BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(out));
writer.write(serverReceived);
writer.newLine();
writer.flush();
sock.shutdownOutput();
BufferedReader reader = new BufferedReader(new
InputStreamReader(in));
StringBuffer buffer = new StringBuffer();
String line;
while ((line = reader.readLine()) != null) {
   buffer.append(line);
}
in.close();
out.close();
sock.close();
serverSent = buffer.toString();
return true;
```

Figura 3.29 *- Metodo connect()*

Infine `connect()` ritorna un `boolean` che assume il valore `true` se la connessione e la richiesta sono andate a buon fine e `false` altrimenti.

L'ultimo metodo che vedremo è `getQuery(String)` che riceve come parametro lo *stream output* `serverSent` e restituisce una stringa che rappresenta l'espansione della *query*.

A causa della lunghezza e della complessità del metodo, ci limiteremo solamente a descrivere la filosofia di espansione scelta, distinguendo il caso di una richiesta monolinguale, da una multilinguale.

Per poter estrarre i dati aggiunti dal *CLIR server*, occorre parsare l'XML di output con *jDom*.

L'operazione preliminare è quella di individuare tutti i nodi rilevanti per il nostro obiettivo: andiamo perciò ad estrarre i tag **`<named_entities>`** e **`<nouns>`**. All'interno del primo nodo troveremo tutti i termini della *query* a cui è stato possibile assegnare una categoria semantica.

Nel CLIR esistono quattro categorie principali:

- `OrganizationType`
- `TimeType`
- `PersonType`
- `PlaceType`

Ognuna di queste dovrà essere necessariamente mappata nelle categorie presenti all'interno della collezione dei *repository ANTS*. Lo scopo è quello di arricchire la *query* di partenza con termini più specifici, del tipo *nome_field:termine*, per affinare i risultati di una possibile ricerca. La mappatura che è stata applicata, segue il seguente schema:

- `OrganizationType`
  - `company`
  - `organization`
  - `organisation`
  - `gov`
  - `airline`
  - `authority`
- `TimeType`
  - `eTime`
  - `sTime`
  - `time`
  - `edition`

- `PersonType`
  - `person`
  - `persona`
  - `famiglia`
- `PlaceType`
  - `citta`
  - `citta_italian`
  - `citta_italiana`
  - `continente`
  - `location`
  - `paese`
  - `regione`

Questo sta a significare che nel caso in cui venga trovato un termine *Iraq* che dal CLIR viene etichettato come `PlaceType`, alla *query* di partenza verranno aggiunti anche i criteri di ricerca

- `citta:Iraq`
- `citta_italian:Iraq`
- `citta_italiana:Iraq`
- `continente:Iraq`
- `location:Iraq`
- `paese:Iraq`
- `regione:Iraq`

separati dall'operatore booleano `OR`.

La *query* assumerà perciò la seguente forma:

```
citta:Iraq OR citta_italian:Iraq OR citta_italiana:Iraq OR
                    continente:Iraq...
```

e così via.

Quello che abbiamo visto, è l'operazione comune effettuata sia in una richiesta monolinguale, sia in una multilinguale. Nel primo caso alla *query* digitata dall'utente verranno aggiunte le espansioni semantiche e la forma finale sarà del tipo:

```
user_query OR (semantic_query)
```

Nel caso multilinguale, questa forma, in cui compare anche la `user_query` (o *query* dell'utente), non sarà possibile, a causa dell'incompatibilità tra linguaggio

source e linguaggio *target*. La *query* dovrà essere perciò necessariamente ampliata con le traduzioni dei termini.

Per realizzare ciò, viene parsato anche il nodo **<nouns>**, in cui possono essere individuate le traduzioni di ogni termine. La scelta della traduzione migliore viene effettuata in base all'attributo `best` del tag **<translations>** *(fig. 3.26)* e selezionando il primo nella lista delle possibili alternative. Avremo perciò una *query* del tipo:

```
semantic_query OR traduzione1 OR traduzione2 OR...
```

in cui non compare la `user_query`.

L'operazione finale è quella di sostituire tutti i termini della `semantic_query` con le traduzioni trovate, per completare il passaggio da linguaggio *source* a linguaggio *target*.

A titolo di esempio, mostriamo la trasformazione della *query* in *figura 3.26*:

```
guerra sanguinosa in Iraq →     (citta:Iraq OR citta_italiana:Iraq OR
                                continente:Iraq OR location:Iraq OR
                                paese:Iraq OR regione:Iraq OR war OR
                                Iraq)
```

4 Conclusioni

Nel presente lavoro si è affrontato il problema dell'indicizzazione e del recupero di dati multimediali, al cui scopo è stato condotto uno studio dell'*Information Retrieval* e delle sue tecniche e scelte implementative. L'obiettivo è stato quello di comprendere in che modo ed in quale misura gli strumenti implementati potessero semplificare e velocizzare la ricerca dell'utente.

Il mio lavoro è passato attraverso varie fasi di studio:

- Studio delle discipline di gestione dei dati, con particolare attenzione agli *inverted indexes* ed al *Vector Space Model*
- Analisi strutturale dei contenuti multimediali di interesse
- Produzione di una nuova struttura di rappresentazione dei dati, che potesse essere valida, oltre che per i programmi *ANTS*, anche per differenti tipi di dato (es. *YouTube* [24])
- Produzione di *tool* di *indexing* (comprensivo di *parser* dei *repository ANTS*)
- Produzione di *tool* di *retrieval*
- Gestione dei processi di interazione *Uomo - Macchina*

Come prima cosa, la scelta di creare un nuovo modello di rappresentazione dei dati, detto *modello standard*, ha permesso di ottenere uno strumento che, oltre a consentire il riutilizzo dello stesso in altri domini applicativi, rende l'intero sistema flessibile e modulare. Infatti, volendo cambiare la tipologia di dati da

indicizzare, basterà utilizzare un *parser* differente, lasciando così invariato il resto del sistema.

L'implementazione della componente di indicizzazione si è rivelata un'ottima soluzione all'archiviazione dei dati, ottenendo buoni risultati di successo per la fase di *indexing* di 7 ore e 40 minuti di telegiornali RAI: questo conferma la robustezza e la applicabilità del sistema realizzato.

Dal punto di vista implementativo, la scelta di usufruire della tecnologia di *Lucene* ha giovato ai fini di una buona produzione del *front-end*, soprattutto perché il *modello standard* si rispecchiava benissimo nella struttura della collezione di *Lucene*.

La componente di ricerca vede in *IndexSearcherWA* la sua realizzazione più completa. Vediamo i risultati ottenuti con l'implementazione di quest'applicazione.

Il servizio di un *retrieval* multilinguale e sensibile ai termini semantici è stato possibile solo grazie all'integrazione del *CLIR server*, che ha come obiettivo quello di esplodere la *query* utente. In questo modo, i semplici termini inseriti dall'utente, subiscono dei processi di analisi finalizzati a:

- comprendere la categoria semantica di ogni termine;
- comprendere la categoria semantica dell'intera *query;*
- fornire una serie di traduzioni, consistentemente con le due operazioni precedenti.

Il risultato è una ricerca più precisa con documenti restituiti più consistenti con l'*information need*.

Il servizio *IndexSearcherWA* offre anche una visualizzazione completa dei documenti, associando le informazioni testuali (trascrizioni e *metadati*) con quelle multimediali (audio/video): è infatti possibile consultare il testo di un documento restituito, piuttosto che il telegiornale di appartenenza o il canale di trasmissione, per poi vedere il frammento di video al quale la notizia fa riferimento.

Il servizio di *import* di nuovi telegiornali da *filesystem* locale completa l'applicazione, integrando così anche la componente di indicizzazione. I processi di ricerca e di *indexing* vengono comunque mantenuti separati, questo per non influire sull'efficienza dei due servizi.

Concludendo, il mio lavoro di sviluppo della piattaforma di IR su dati multimediali ha prodotto una buona alternativa di *indexing* e recupero di contenuti multimediali che operano su particolari archivi multimediali.

La peculiarità dei *repository* gestiti, fa si che una comparazione con modelli già

presenti in letteratura sia complessa, se non impossibile. Ciò non toglie che il modello qui prodotto possa essere completamente standardizzato con espansioni minime, come è indicato negli sviluppi futuri descritti nel prossimo paragrafo.

4.1 Sviluppi futuri

Il lavoro fin qui svolto potrebbe essere ulteriormente migliorato, prestando attenzione ai problemi riscontrati e ad una migliore gestione del *front-end*.

Avendo a disposizione nuove strutture dati (le pagine di un quotidiano online, una collezione di *e-book*...), si potrebbe pensare di modificare ulteriormente il modello standard, in modo da renderlo un modello universale e creare così una collezione variegata in cui filmati, documenti di testo, pagine web siano indicizzati con lo stesso modello dati.

L'integrazione di nuovi *parser*, implementati ad hoc su ulteriori strutture dati, potrebbe semplificare all'amministratore la fase di *indexing*. Infatti, la possibilità di scegliere il *parser* adeguato durante il processo di *import* di un nuovo documento da indicizzare, andrebbe ad eliminare eventuali modifiche del codice.

Sfruttando la dinamicità dell'architettura utilizzata, potrebbe essere utile mutare la presentazione del *front-end*, adattandolo al tipo di contenuto cercato. Questa accortezza aumenterebbe le capacità di interazione dell'utente con l'applicazione.

Anche il problema della visualizzazione del contenuto multimediale nella maggior parte dei *browser web* in circolazione potrebbe essere risolto adottando *plugin* diversi.

Il possibile sviluppo futuro di queste migliorie e di queste soluzioni possono rendere l'applicazione implementata, un prodotto finito, pronto per essere utilizzato dagli amministratori ed a servizio dell'utente.

5 Appendice: una tipica sessione utente

5.1 Front-end

La *figura 5.1* mostra l'interfaccia di presentazione dell'applicazione *IndexSearcherWA*.

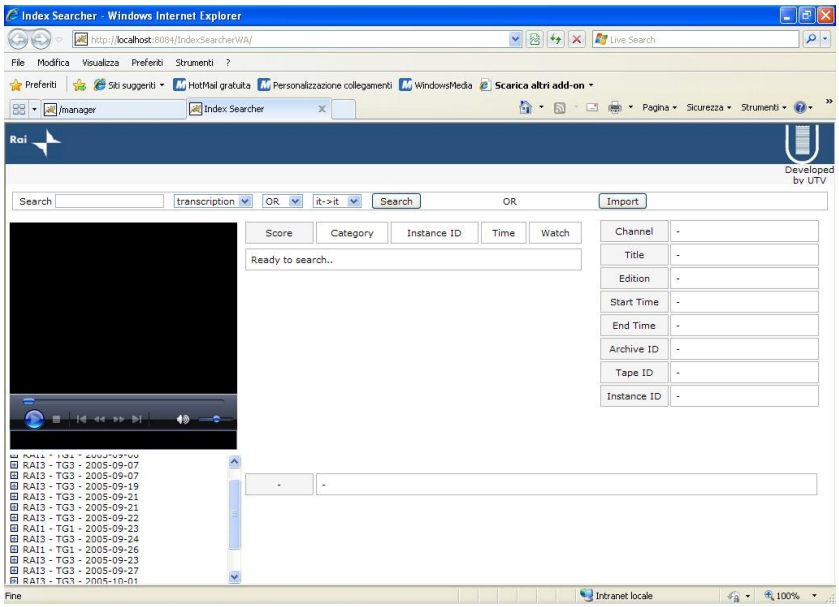


Figura 5.1 *- IndexSearcherWA: schermata principale*

Essendo questa una *web application*, l'unico modo per poter usufruire dei servizi che offre è quello di utilizzare un browser. Nonostante l'applicazione sia stata sviluppata interamente in ambiente Unix, la scelta di utilizzare *Internet Explorer* è stata obbligata dal fatto che, per poter visualizzare correttamente i contenuti multimediali attraverso *Windows Media Player*, è necessario utilizzare *ActiveX* e plugin proprietari *Microsoft*. In ogni caso, l'utilizzo di tutti gli altri servizi è garantito anche in browser differenti.

La pagina *index* è composta dal frame dedicato alla ricerca e all'import di nuovi telegiornali (*in alto*), dal frame in cui vengono visualizzati i risultati (*al centro*), da quello per i *metadati* (*a destra*), il frame per il player (*a sinistra*), il menu della collezione (*in basso a sinistra*) e il frame per visualizzare le trascrizioni della singola notizia (*in basso a destra*).

Nel momento in cui viene effettuata la ricerca, il sistema avvisa, come prima cosa, se il *CLIR server* è attivo o meno *(fig. 5.2)*.

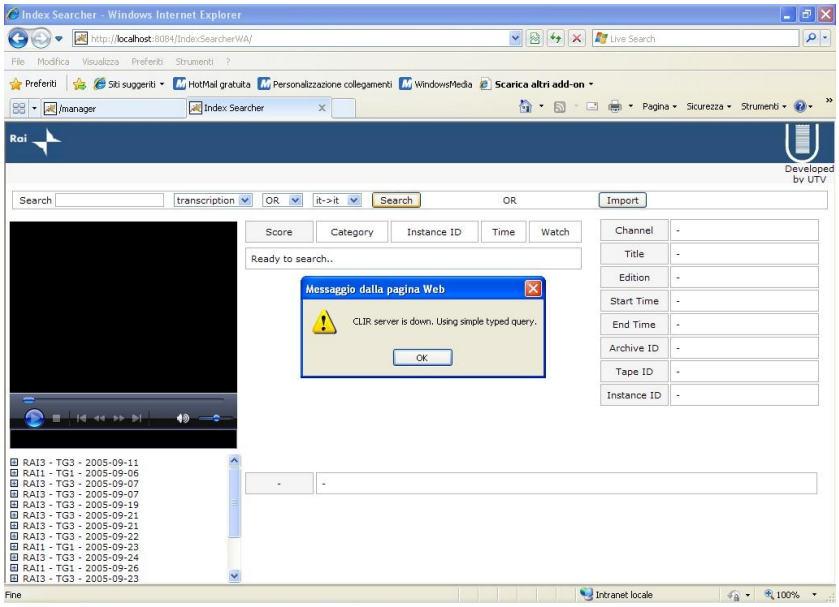


Figura 5.2 *- IndexSearcherWA: interrogazione con CLIR server non attivo*

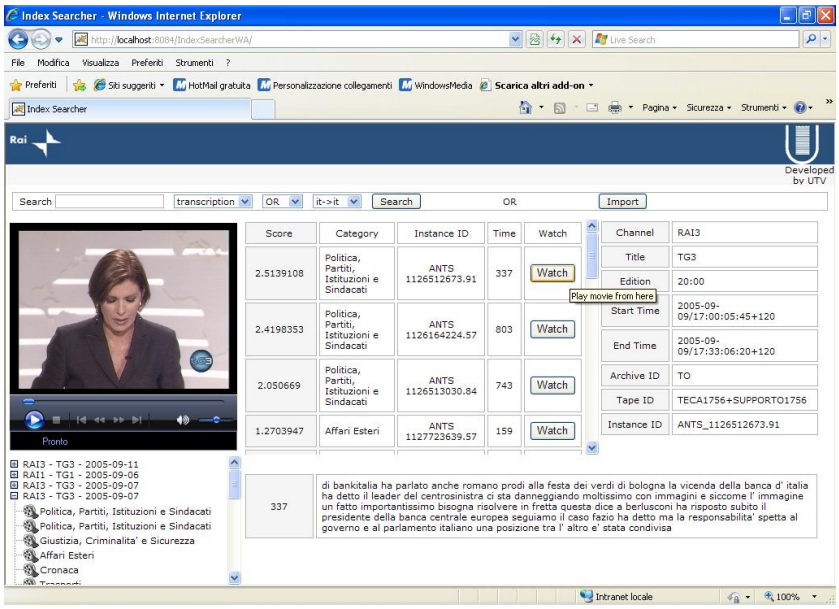


Figura 5.3 *- IndexSearcherWA: visualizzazione dei risultati e del filmato*

Nel caso in cui esso non sia attivo, la ricerca procede comunque, utilizzando la *query* dell'utente senza le informazioni prodotte dal *CLIR server*. È così possibile cliccare su un risultato per visualizzare *metadati* e trascrizioni, o sul tasto *Watch* che abilita lo *streaming* del filmato corrispondente, nell'istante in cui la notizia occorre*(fig. 5.3)*.

5.2 Interrogazione in italiano

Nel caso in cui si voglia effettuare una interrogazione in italiano, dovrà essere selezionata, nella terza *combo box*, la voce *it -> it*. In questo modo il sistema adotterà la soluzione monolinguale per espandere la *query* dell'utente.

Supponiamo che l'utente inserisca una *query* del tipo:

Berlusconi al parlamento sulla missione di guerra in Iraq

Il sistema analizzerà il testo e costruirà un'espressione arricchita dalle informazioni semantiche:

Berlusconi al parlamento sulla missione di guerra in Iraq OR (person:Berlusconi OR persona:Berlusconi OR famiglia:Berlusconi OR Iraq)

I risultati saranno cercati utilizzando la seconda forma e visualizzati nello stesso modo di una sessione di ricerca con *CLIR server* inattivo *(fig. 5.3)*.

Inoltre, il sistema si preoccuperà di stampare a video eventuali messaggi di errore recapitati dal CLIR. Come mostra la *figura 5.4*, il risultato che ha ottenuto il punteggio più elevato è la news di "Affari Esteri" del telegiornale con identificatore *ANTS_1126513030.84* che occorre nel secondo 8.

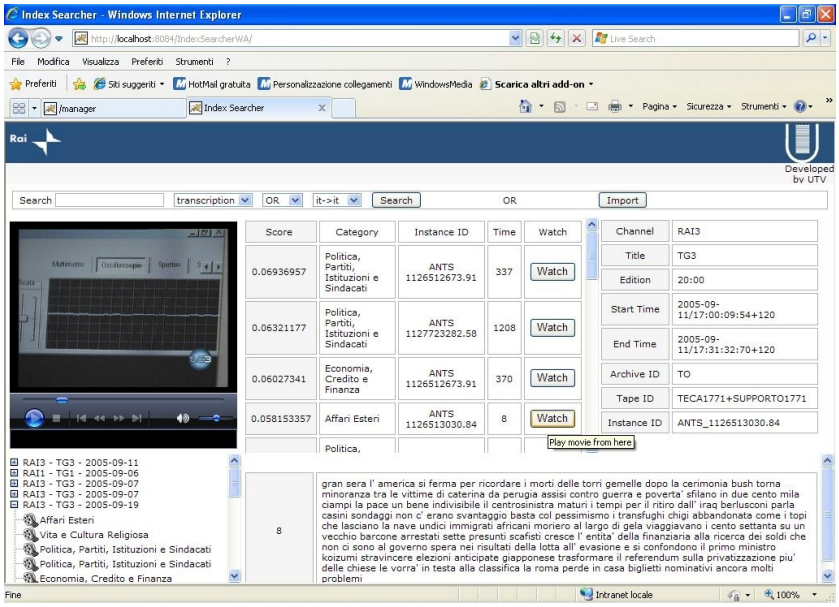


Figura 5.4 *- IndexSearcherWA: interrogazione in italiano*

Vedremo, nel paragrafo seguente, come i risultati ottenuti siano approssimativamente consistenti con quelli ottenuti da un'interrogazione in inglese.

5.3 Interrogazione in inglese

Per effettuare una interrogazione in inglese, cambiamo il valore della *combo box* in *en -> it* e proviamo ad utilizzare una *query* simile a quella utilizzata nella ricerca in italiano:

Berlusconi in the parliament about Irak war mission

Il sistema, dopo aver analizzato la *query*, effettua la ricerca sulla base di un'espressione della forma:

(person:Berlusconi OR persona:Berlusconi OR famiglia:Berlusconi OR citta:Irak OR citta_italiana:Irak OR continente:Irak OR location:Irak OR paese:Irak OR regione:Irak OR parlamento OR lotta OR missione OR Berlusconi OR Irak)

ottenendo in quarta posizione, la news che nell'interrogazione in italiano deteneva il punteggio più alto *(fig. 5.5)*.

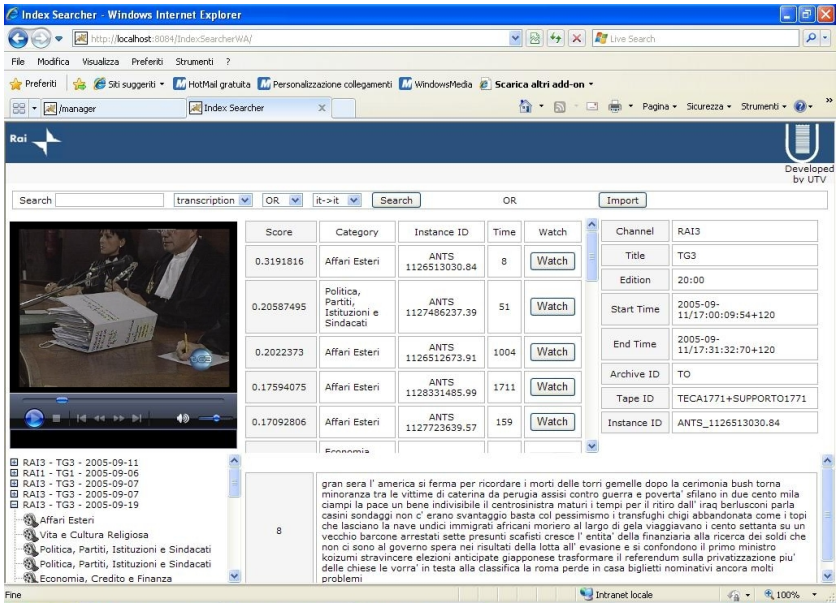


Figura 5.5 *- IndexSearcherWA: interrogazione in inglese*

5.4 Import di nuovi telegiornali

Come abbiamo detto, *IndexSearcherWA* offre all'utente anche la possibilità di importare nuovi telegiornali nella collezione. La procedura prevede l'inserimento del path in cui è possibile trovare il *repository ANTS (fig. 5.6)*.

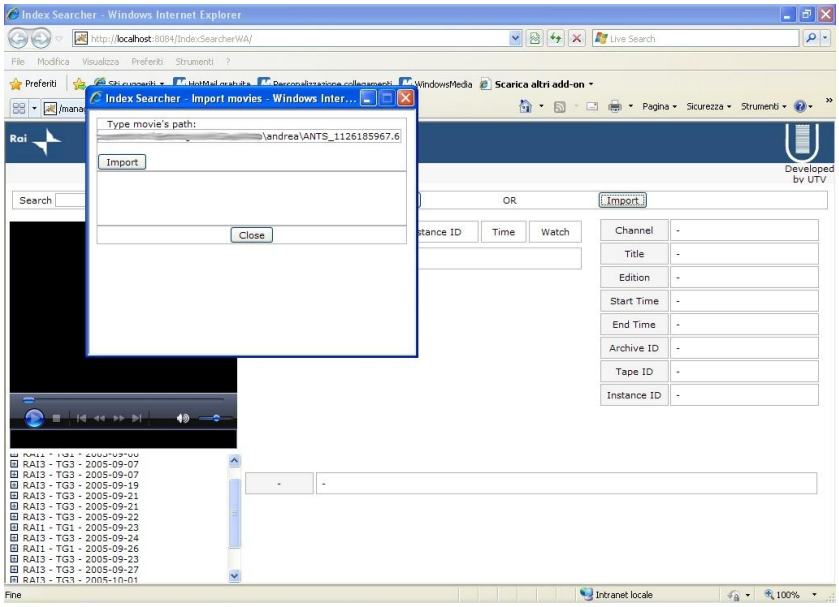


Figura 5.6 *- IndexSearcherWA: inserimento del path*

Dopo aver elaborato le operazioni necessarie alla buona riuscita dell'*import*, la pagina verrà ricaricata, mostrando l'*id* del telegiornale e il numero delle news contenute nel caso in cui il processo si sia concluso con successo *(fig. 5.7)*, altrimenti visualizzerà un messaggio con l'errore riscontrato.

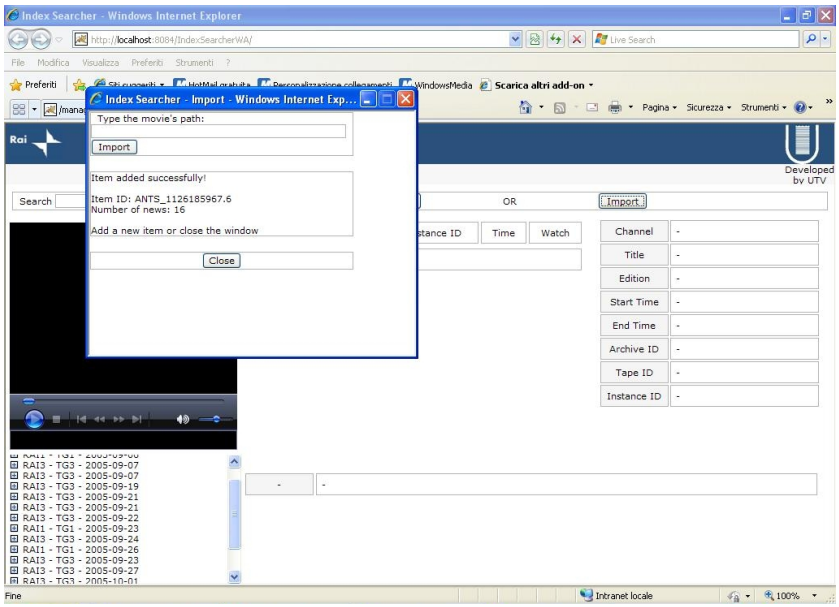


Figura 5.7 *- IndexSearcherWA: successo dell'import*

Bibliografia

[1] Calvin E. Mooers. 1950. *Coding, information retrieval, and the rapid selector*

[2] Douglas W. Oard and Bonnie J. Dorr. 1996. *A survey of multilingual text retrieval*. University of Maryland, College Park, MD, USA

[3] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze. 2009. *An Introduction to Information Retrieval*. Cambridge University Press, Cambridge, England

[4] Gerard Salton. 1979. *Mathematics and Information Retrieval*. Department of Computer Science, Cornell University, Ithaca, NY, USA

[5] Alistair Moffat and Justin Zobel. 1996. *Self-indexing inverted files for fast text retrieval*. ACM Press, New York, NY, USA

[6] Justin Zobel and Alistair Moffat. 2006. *Inverted files for text search engines*. ACM Computing Surveys

[7] Jetty Web Server, http://jetty.codehaus.org/jetty/

[8] Apache Tomcat, http://tomcat.apache.org/

[9] Sun GlassFish, https://glassfish.dev.java.net/

[10] Java Servlet, http://java.sun.com/products/servlet/

[11] JavaServer Pages, http://java.sun.com/products/jsp/

[12] Common Gateway Interface, http://www.w3.org/CGI/

[13] Microsoft ASP.NET, http://www.asp.net/

[14] Microsoft Internet Explorer, http://www.microsoft.com/italy/windows/internet-explorer/

[15] Zettair, http://www.seg.rmit.edu.au/zettair/

[16] Wumpus Search Engine, http://www.wumpus-search.org/

[17] Terrier IR Platform, http://terrier.org/

[18] Apache Lucene, http://lucene.apache.org/

[19] Apache Software Foundation, http://www.apache.org/

[20] Apache Lucene - File Formats, http://lucene.apache.org/java/3_0_1/fileformats.html

[21] Apache Lucene - Scoring, http://lucene.apache.org/java/3_0_1/scoring.html

[22] Sun Microsystem, http://www.oracle.com/

[23] Roberto Basili, Marco Cammisa, Alessandro Moschitti, Daniele Pighin and Alfredo Serafini. 2007. *Cross-linguistic IE tools for metadata discovery*. Università di Tor Vergata, Rome, Italy

[24] YouTube, http://www.youtube.com/