UNIVERSITA' DEGLI STUDI DI NAPOLI
FEDERICO II

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

DIPARTIMENTO DI INFORMATICA E SISTEMISTICA

TESI DI LAUREA IN SISTEMI INFORMATIVI

REALIZZAZIONE DI UN MOTORE DI RICERCA SEMANTICO

Relatore
Ch.mo Prof. Antonio d'Acierno

Co-Relatore
Ing. Antonio Maria Rinaldi

Candidato
Luigi Amalfitano
041/1744

Anno Accademico 2002/2003

Non chi comincia
ma quel che persevera

veliero Amerigo Vespucci, motto

Indice

Introduzione VI

Organizzazione VIII

Capitolo 1 Definizione del Progetto 1

1.1 Il sistema 2

1.2 Casi d'uso 3

User 3

Admin 4

1.3 Specifiche 5

Capirolo 2 Ontologia 7

2.1 Definizione di Ontologia 8

2.2 Modello formale 9

Domini, Concetti e Parole 9

2.3 Semantic Network 13

2.4 La metrica 17

Posizione mediatae Grado Sintattico 17

Contributo Semantico 19

Voto Complessivo 22

Capitolo 3 Il Sistema 23

3.1 Architettura 24

Search Engine Wrapper 25

Web Spider 25

Document Preprocessor 26

Miner 27

3.2 Sistema Distribuito 29

Comunicazione tra gli utenti e il sistema 33
Diagramma dei componenti e di deployment 34

3.3 Base Dati del Sistema 37
Repository 37
MotoriDiRicerca 40
Query 42
Link 43
Documenti e DocumentiPreProcessati 43
Stato 44
Semantic Knowledge Base 45

Capitolo 4 Tecnologie Utilizzate 46

4.1 Strumenti di sviluppo 47

4.2 Linguaggio di programmazione 49

4.3 Oggetti Remoti 50
Il ruolo del Client e del Server 50
Remote Method Invocation 53
Stub e Marshal dei parametri 54
Caricamento di classe dinamico 56

4.4 Collegamento al DataBase: JDBC 57
La struttura di JDBC 58
Utilizzo tipo di JDBC 62

4.5 DBMS 63

4.6 Interazione con il sistema 64

Capitolo 5 Sviluppo del Sistema 65

5.1 Package Util 66
La classe Cronometro 67
La classe webPage 68
La classe DBMS 70

5.2 Server Spider 72
La classe Link 74
La classe MotoriDiRicerca 77
La classe Query 79

5.3 Server Preprocessor 83
La classe Documento 85
La classe PreProcessa 88
La classe QueryPrePro 90
Gestione dei Thread
PreProcessa e tempificazione 94

5.4 Semantic Network 95
Definizione della Semantic Network 97
La classe StopWords 99
La classe Testo 101

5.5 Server Miner 102
La classe DocumentoPreProcessato 104
La classe MinerPages 106
La classe QueryMiner 108
Gestione dei Thread
MinerPages e tempificazione 112

Capitolo 6 Interfacce 114

6.1 Architettura RMI del sistema 115

6.2 L'interfaccia utente 123
Nuova Query 124
Visualizzazione dei Risultati 127
Modifica e Inserimento
di nuovi motori di ricerca 128

Capitolo 7 Sperimentazione 131

7.1 Ambiente utilizzato 132
Attivazione del sistema 133

7.2 Prove effettuate 135
Considerazioni Finali 136
Query Morandi – musica 137
Query Zucchero – musica 139
Query Amerigo Vespucci – mare 142
Query Pesca – mare 144

Bibliografia 147

Introduzione

Il World Wide Web è diventato progressivamente una vasta risorsa di informazioni. A causa della complessità nell'organizzazione dei dati e della quantità di materiale presente, la ricerca sul WEB di informazioni davvero utili per un utente è diventata decisamente complessa: anche se un motore di ricerca Web assiste nella scoperta di risorse, esso è lontano dal soddisfare, per la sua scarsa precisione, le richieste di una ricerca mirata. Infatti, trovare le informazioni richieste nei risultati di un motore di ricerca si rivela fruttuoso solo in presenza di argomenti di una certa notorietà e importanza, e di query molto precise; negli altri casi questo lavoro può implicare una considerevole perdita di tempo che un utente può pagare in termini di costi di collegamento alla rete.

Questo avviene perché i motori di ricerca tradizionali effettuano ricerche di tipo sintattico: essi restituiscono le pagine che contengono le keywords presenti nelle query degli utenti, indipendentemente dal contesto in cui esse sono utilizzate.

Se ciò da un lato è conveniente in termini di velocità di reperimento delle pagine e restituzione dei risultati, dall'altro lato porta spesso a risultati errati o imprecisi, dato che vengono restituite molte pagine non attinenti al contesto della query dell'utente. Ad esempio, un utente che voglia cercare pagine relative alla pesca, inserendo in un motore come keyword la parola "pesca", oltre a trovare pagine attinenti al dominio ittico troverà sicuramente pagine inerenti al dominio ortofrutticolo.

Inoltre, sempre più sono i motori di ricerca che ordinano i documenti web secondo delle politiche commerciali: la registrazione a pagamento di un sito in un motore di ricerca offre una serie di vantaggi, primo fra tutti la certezza che il sito sia presente in te-

sta all'elenco o nelle prime posizioni, anche se tratta marginalmente un determinato argomento.

Sarebbe conveniente avere a disposizione un sistema in grado di capire di cosa parla una pagina, valutando la sua attinenza con i domini di interesse per l'utente. Una ricerca di tale tipo è detta semantica in quanto non restituisce semplicemente pagine che contengono le keywords, ma pagine che hanno anche un contenuto semantico aderente al dominio desiderato dall'utente.

Il nostro scopo è stato quello di realizzare un motore di ricerca semantico, partendo dalla progettazione ad alto livello fino ad arrivare all'implementazione e al testing finale. Il sistema è stato realizzato secondo metodi ingegneristici usando l'Unified Modeling Language – UML per esprimere le caratteristiche e il funzionamento del progetto stesso. Inoltre, la base dati utilizzata dal motore di ricerca è stata dapprima sviluppata da un punto di vista concettuale, utilizzando il modello E-R, e, successivamente, a livello logico, con il modello relazionale.

Data la sua complessità il sistema sarà presentato a "livelli" di dettaglio successivi: ciascun livello porrà l'attenzione su particolari aspetti del motore di ricerca.

Organizzazione

Ciascun capitolo della tesi affronta un argomento ben preciso utilizzando un determinato punto di vista e ad un livello di dettaglio sempre maggiore:

- Capitolo 1

  Vengono definite le caratteristiche e le specifiche del sistema “Motore di Ricerca Semantico”. Si individuano gli utenti e il modo in cui interagiscono con il sistema.

- Capitolo 2

  Viene presentato il modello formale alla base dell’analisi semantica adottata dal sistema: si espone la metrica utilizzata.

- Capitolo 3

  Nella prima parte del capitolo viene presentata l’architettura del sistema a livello concettuale. Nella seconda parte il punto di vista sarà di specifica a livello di interfaccia. Nella terza parte si espone lo schema concettuale della base dati.

- Capitolo 4

  In tale capito si presentano e giustificano le scelte tecnologiche adottate.

- Capitolo 5

  Ora il punto di vista è di implementazione. Il livello di dettaglio è medio-alto. Non si fa riferimento al codice, tuttavia viene evidenziando il funzionamento degli oggetti e il modo in cui interagiscono.

- Capitolo 6

  Si espone il modo in cui è stata realizzata la comunicazione con il sistema e le caratteristiche dell'interfaccia utente. Il punto di vista è sempre di implementazione con un livello di dettaglio medio-alto.

- Capitolo 7

  Si presentano alcuni risultati sperimentali, mettendo in luce l'efficacia e l'efficienza del sistema.

- Allegato

  In allegato al presente volume è riportata la documentazione API relativa a tutte le classi realizzate e lo schema fisico della base dati.

Capitolo 1

Definizione del Progetto

1.1 Il sistema
1.2 Casi d'uso
1.3 Specifiche

I motori di ricerca web sono in realtà dei grandi cataloghi che hanno la capacità di archiviare le informazioni relative alle pagine web prima che queste vengano richieste. La loro efficienza è valutata nei tempi necessari a fornire un set di pagine all'utente che effettua un'interrogazione.

Tale caratteristica verrà sfruttata dal motore di ricerca semantico per reperire le pagine dal web, filtrarne il contenuto ed analizzarlo; la ricerca effettuata dall'utente richiederà un'elaborazione ben più complessa di quella di un normale motore. Sarà necessario, quindi, un'azione di controllo in linea del sistema che permetta di gestire l'efficienza ed aumentarne l'efficacia.

Il motore di ricerca semantico dovrà garantire un tempo di risposta accettabile, che spinga l'utente ad utilizzarlo realmente. Nel progetto bisognerà tenere presente che il sistema dovrà essere utilizzato non per fare delle sporadiche ed isolate ricerche semantiche, ma per un utilizzo massiccio e continuato da parte di comuni utenti internet.

1.1 IL SISTEMA

Con il termine ricerca sintattica dei documenti si intende una ricerca che individui una serie di documenti che contengano l'insieme di parole (interrogazione, query) definite dall'utente. Una ricerca semantica, invece, parte da una ricerca sintattica e analizza i documenti da un punto di vista semantico al fine di attribuirne un significato che soddisfi l'interrogazione effettuata dall'utente. Quindi, una query semantica è composta dalle seguenti parti:

- un insieme di vocaboli da ricercare;
- un dominio di appartenenza;

Per esempio, l'utente che vuole informazioni relative al cantante zucchero, dovrà indicare zucchero come vocabolo e musica come dominio.

Il motore di ricerca semantico dovrà quindi effettuare le seguenti operazioni:

- Spidering:

  ovvero ricercare documenti web che contengano l'insieme dei vocaboli indicati dall'utente. Tale operazione viene demandata ai motori di ricerca tradizionali poiché la effettuano in maniera efficiente.

- Preprocessing:

  eliminare dal documento web gli elementi che non rappresentano informazione: codici html, script, applet, etc … .

- Mining:

  analizzare il contenuto del documento da un punto di vista semantico attribuendone un voto.

- Reporting:

  riportare l'insieme dei documenti trovati, ordinati per voto.

Tali operazioni dovranno essere effettuate per ciascuna interrogazione.

1.2 CASI D'USO

Il sistema dovrà interagire con due tipologie di utenti:

- User:

  e' l'utente che può effettuare la ricerca semantica, sottoponendo una query al motore di ricerca, e può visualizzare l'elenco dei documenti trovati per ciascuna query.

- Admin:

  è l'amministratore (eventualmente anche più di uno) del sistema. Può agire sui parametri di configurazione, sulle caratteristiche dell'elaborazione sintattica e semantica del motore di ricerca.

USER

L'utente generico interagisce con il motore di ricerca come indicato dal seguente caso d'uso:

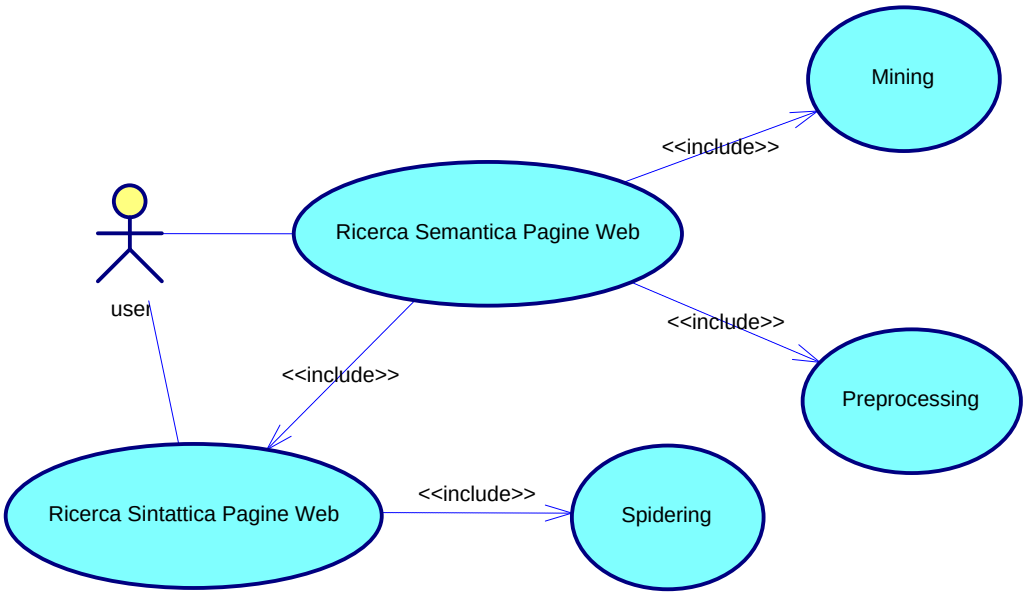

Figura 1.1: caso d'uso relativo all'utente generico

L'utente generico, nell'utilizzare il sistema, dovrà sottoporre l'interrogazione così come è stata definita nel paragrafo 1.1.

Poiché lo spidering richiede l'utilizzo di diversi motori di ricerca, l'utente potrà selezionare quali motori di ricerca utilizzare e che peso relativo assegnare a ciascuno di essi: tali informazioni serviranno per assegnare un peso sintattico ai documenti. Inoltre, potrà avere un controllo sui parametri di ricerca relativi all'interrogazione formulata. Per maggiori dettagli si rimanda ai capitoli successivi.

L'elenco dei documenti relativi ad una data interrogazione sarà disponibile in maniera permanente. Questo per dare la possibilità all'utente di visualizzare anche successivamente i risultati ottenuti e anche per utilizzarli per ricerche simili. Infatti, è difficile che un documento web possa modificare in tempi brevi il suo contenuto semantico (se una pagina web parla del cantante zucchero, difficilmente potrà parlare di zucchero come alimento).

ADMIN

Il sistema dovrà permettere di intervenire sull'elenco dei motori di ricerca a disposizione degli utenti e sulla rete semantica.

L'utente Admin, oltre a poter eseguire normali ricerche, come un utente generico, potrà interagire con il sistema come indicato dal seguente caso d'uso:

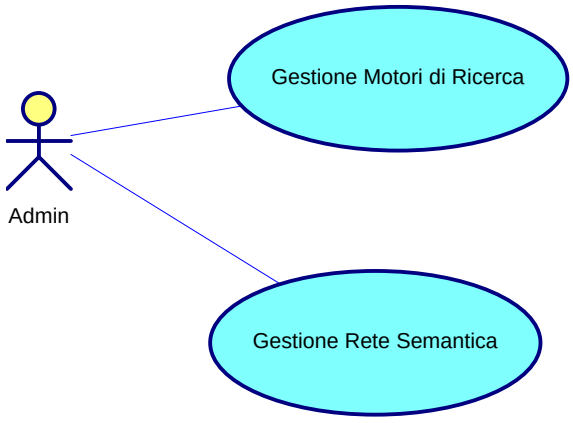

Figura 1.2: caso d'uso relativo all'amministratore del sistema

Il compito dell'amministratore sarà quello di controllare il sistema al fine di aumentarne le performance. Questo può essere fatto analizzando il comportamento dei motori di ricerca tradizionali per inserirli con un peso opportuno all'interno dell'elenco, in modo che, all'utente, sia sempre disponibile il meglio della ricerca sintattica.

Potrà aggiungere vocaboli al dizionario e riconfigurare la rete semantica per domini specifici.

L'amministratore valuterà la convenienza di utilizzare una o più risorse hardware che possano far aumentare le prestazioni del sistema.

1.3 SPECIFICHE

Da quanto esposto si possono elencare le specifiche del sistema "motore di ricerca semantico":

- La ricerca semantica deve essere:
  - Efficace: il set di pagine deve essere semanticamente valido e l'utente deve poter valutare l'efficacia controllando anche le pagine che sono state scartate.
  - Efficiente: La ricerca semantica dei documenti deve avvenire nel minor tempo possibile. Consideriamo una classica ricerca sul Web: sia t il tempo che un normale motore di ricerca impiega per restituire le pagine e T il tempo impiegato dall'utente per individuare i documenti voluti, di solito si ha che T>>t; la ricerca da parte del nostro sistema deve avvenire in t'<T. Osserviamo che la ricerca semantica è ben più complessa di quella sintattica, quindi in generale si ha che t'>t, tuttavia l'efficienza del sistema deve poter essere evidente all'utente, che deve poter risparmiare tempo e notare la differenza con un normale motore di ricerca.
- Accessibile:
  - Il sistema deve essere accessibile a tutti mediante internet e l'utilizzo di un normale browser.

- Concorrenza nelle richieste:
    - Al sistema possono pervenire contemporaneamente più richieste: deve essere in grado di gestirle senza far degradare l'efficienza.

- Configurabile e riconfigurabile:
    - si deve poter intervenire, tramite internet ed un browser, in maniera agevole, sui meccanismi di ricerca e di analisi delle pagine senza dover sospendere l'attività del sistema.

- Indipendenza del sistema sia software che hardware:
    - Il sistema dovrà essere progettato attenendosi agli standard, con la possibilità di poter modificare il linguaggio di programmazione utilizzato. Si dovrà, inoltre, per quanto possibile, avere un'indipendenza dal sistema operativo e dai supporti software utilizzati (server DB). In ultimo si dovrà garantire l'indipendenza dalla piattaforma hardware usata.

Nel progettare il sistema sono state soddisfatte tutte le specifiche elencate ed il sistema è stato sottoposto ad un testing iniziale. E' pronto per essere attivato on-line e messo a disposizione degli utenti internet. In questo modo sarà possibile avere una soddisfacente fase di beta-testing a valle della quale il sistema potrà funzionare a pieno regime.

Capitolo 2

Ontologia

2.1 Definizione di Ontologia
2.2 Modello formale
2.3 Semantic Network
2.4 La metrica

Nel corrente capitolo si affronta il problema di individuare un modello che sia in grado di rappresentare un linguaggio naturale, in molti casi ambiguo. Viene presentata, quindi, la base teorico-formale considerata nello sviluppo del sistema.

Partendo da elementi come dominio, concetto e parola, sfruttando la teoria della probabilità, si sviluppa un grafo orientato (rete semantica) che possa esprimere il contenuto semantico di un dato set di vocaboli.

Il peso assegnato alla pagina web sarà funzione di un'analisi sintattica e di una semantica, che interesseranno, in modo diverso, le parti più importanti del documento web: titolo, descrizione, parole chiavi e contenuto.

2.1 DEFINIZIONE DI ONTOLOGIA

Un'ontologia è la specificazione esplicita di una concettualizzazione[11]. E' possibile definirla come una descrizione dei concetti e delle relazioni che intercorrono tra di essi. Essa permette di lavorare con un insieme strutturato di concetti, dove le relazioni risultino chiare e, soprattutto, significative a livello semantico. Una ontologia può essere specificata informalmente mediante proposizioni in linguaggio naturale. Una formale è, invece, specificata da una raccolta di nomi per i concetti che si vogliono specificare e dalle relazioni che intercorrono tra di loro.

Le ontologie sono spesso considerate alla stregua delle gerarchie tassonomiche di classi, ma, le prime, nella visione di T. Gruber[11], non si limitano a dare definizioni conservative, ovvero definizioni nel senso tradizionale di introdurre terminologia senza aggiungere conoscenza a proposito del mondo. Infatti, per mantenere un sistema "aperto", l'ontologia non è basata su una gerarchia fissa di categorie, ma su un insieme di distinzioni, a partire dalle quali la gerarchia è generata automaticamente. E' importante considerare gli obiettivi di un'ontologia. Se ne progetta una per permettere la condivisione della conoscenza e il suo riutilizzo. Pragmaticamente, si sceglie di scrivere un'ontologia come un insieme strutturato di definizioni di un vocabolario formale. Sebbene questo non sia l'unico modo di specificare una concettualizzazione, esso ha l'interessante proprietà di permettere la condivisione della conoscenza tra software di intelligenza artificiale. E' sostanzialmente un accordo sull'uso di un certo vocabolario (ovvero effettuare interrogazioni e fare affermazioni) in modo che esso sia consistente (ma non completo) rispetto alla teoria specificata dall'ontologia. A partire dalle ontologie, si possono costruire agenti che fanno affidamento su di esse. Le ontologie sono quindi progettare allo scopo di condividere la conoscenza con e tra gli agenti. Non è richiesto comunque che un tale agente risponda a tutte le domande che possono essere formulate nel vocabolario condiviso. Un accordo sull'uso di una ontologia comune è una garanzia di consistenza ma non di completezza rispetto alle interrogazioni e alle affermazioni che possono essere fatte usando il vocabolario definito dall'ontologia stessa.

2.2 MODELLO FORMALE

Consideriamo un insieme di elementi A. Nella *teoria classica degli insiemi* l'appartenenza di ogni elemento x ad A è indicata in maniera certa. Ci sono però dei casi in cui la teoria classica non riesce a classificare in maniera adeguata gli elementi di un sottoinsieme dell'universo del discorso. In questi casi può essere opportuno non indicare la semplice appartenenza di un elemento all'insieme, ma la probabilità che l'elemento appartenga o meno all'insieme: questa probabilità di appartenenza sarà un valore compreso nell'intervallo [0,1] dei numeri reali ed ovviamente l'appartenenza certa si avrà con la probabilità pari ad 1, mentre in corrispondenza della probabilità pari a 0 si avrà la non appartenenza all'insieme.

DOMINI CONCETTI E PAROLE

Iniziamo con lo stabilire con precisione cos'è una lingua e quali sono gli elementi che la caratterizzano.

Definizione 2.2.1: Vocabolo

> *Si definisce vocabolo un insieme di caratteri isolato, colto fuori da un contesto e da ogni legame grammaticale o logico, nella sua individualità lessicale*[12].

Esempio 2.2.1. Alcuni vocaboli sono, ad esempio:

> *riserva, casa, scuola, elicottero, auto...*

I vocaboli sono la parte fondamentale di una lingua.

Definizione 2.2.2: Lingua

> *Si definisce una lingua come l'insieme dei vocaboli utilizzati da un popolo, insieme alle regole che li governano*[12].

Esempio 2.2.2. Alcune possibili lingue possono essere:

ITALIANO, TEDESCO, INGLESE, SPAGNOLO,...

Definizione 2.2.3: Regole sintattiche

Si definiscono regole sintattiche (o sintassi) l'insieme delle norme che studiano le relazioni che i vocaboli hanno in una frase (gruppo di vocaboli)[12].

Definizione 2.2.4: Regole semantiche

Si definiscono regole semantiche (o semantica) l'insieme delle norme che si occupano del significato dei vocaboli e dei cambiamenti di essi[12].

Ci occuperemo di seguito della lingua italiana e delle sue regole sintattiche e semantiche. Definiamo ora cosa è un dominio ed un concetto.

Definizione 2.2.5: Dominio

Detto S l'universo del discorso si definisce dominio D un sottoinsieme di S, ovvero

$$D \subseteq S$$

Esempio 2.2.3. Possibili esempi di domini, nella lingua Italiana, potrebbero essere:

il dominio della MUSICA, della PITTURA, dello SPORT ecc.

Definizione 2.2.6: Concetto

Per concetto linguistico si intende una parola, frase, acronimo o nome ricco di significato, che è stato estratto da componenti non strutturati del testo, incluso blocchi isolati di testo, sommari, intestazioni, paragrafi etc. Esso è definito anche come termine linguistico, mentre ogni documento si riferisce a un gruppo di concetti o termini[13].

Dato un dominio D e detti C_i i concetti in esso definiti si ha che l'unione di tutti i concetti è una copertura per il dominio stesso, ovvero:

$$D = \bigcup_{i=1}^{n} C_i$$

Esempio 2.2.4:

Se consideriamo il dominio della MUSICA, possibili esempi di concetti di tale dominio potrebbero essere il concetto STRUMENTI, il concetto CANTANTI, ecc.

Ogni concetto è costituito da vocaboli della lingua alla quale si ci riferisce. Possiamo però considerare che, i vocaboli, non appartengano ad un concetto necessariamente con una probabilità pari ad 1. Soprattutto nei casi in cui si vuole rappresentare un dominio con un insieme sintetico di concetti, può accedere che alcuni vocaboli facenti parte del dominio, siano inseriti in concetti ai quali non appartengono pienamente. Inoltre un vocabolo può appartenere a più concetti e pare opportuno, in questi casi, immaginare che non appartenga appieno ad entrambi i concetti.

Esempio 2.2.5:

Consideriamo ad esempio i due vocaboli della lingua italiana: basso **e** *fisarmonica. Possiamo immaginare che fisarmonica appartenga al concetto strumento del dominio musicale con una probabilità maggiore rispetto a basso, in quanto quest'ultimo crea maggiori ambiguità.*

Per tale motivo introduciamo il concetto di centralità di un vocabolo rispetto ad un concetto.

Definizione 2.2.7: Centralità

Definiamo centralità la probabilità percentuale di appartenenza del vocabolo lessicale ad un concetto.

Appare evidente che più il valore della centralità è prossimo ad 1 e più è probabile che il vocabolo appartenga al concetto associato.

Definizione 2.3.8: Peso

Definiamo peso il grado con cui un vocabolo lessicale rappresenta il dominio al quale è associato.

Anche il peso è un valore appartenente all'intervallo [0, 1] e più è grande, maggiore è il livello di rappresentazione, da parte del vocabolo, del dominio associato.

Definizione 2.2.9: Parola

Definiamo parola un vocabolo di una lingua, a cui è associato un concetto C, il dominio D a cui il concetto appartiene e due indici centralità e peso, ovvero

< vocabolo lessicale, concetto, dominio, centralità, peso >

Indicato con S sempre l'universo del discorso, è possibile anche definire una funzione F tale che data una parola x ed un concetto C appartenente ad un dominio D, il valore

$$F_C(x) \in [0,1]$$

rappresenta la centralità di x rispetto al concetto C, con la convenzione che se il vocabolo inserito nella parola x non appartenga al concetto C la funzione F restituisca 0.

In maniera analoga alla centralità è possibile definire una funzione P tale che dato una parola x ed un dominio D, il valore

$$P_D(x) \in [0,1]$$

rappresenta il peso di x rispetto al Dominio D.

La definizione di un modello che soddisfi i nostri scopi deve tenere conto, in modo prevalente, della capacità di un dato vocabolo di rappresentare un dominio. Inoltre il modello dovrà essere quanto più "leggero" possibile in modo che possa essere facile da implementare e gestire. Per tali motivi trascureremo l'informazione relativa alla centralità e supporremo che un vocabolo, dato un dominio, sia associato al più ad un unico concetto, quello che meglio riesce ad esprimere. Quindi la *parola* sarà la quadrupla:

< vocabolo, concetto, dominio, peso >

A titolo di esempio si elencano le seguenti parole:

Vocabolo	Concetto	Dominio	Peso
Chitarra	Strumenti	Musica	0.6
Olio	Ricambi	Motori	0.55
Aereo	Veicoli	Turismo	0.7
Bacio	Amore	Sentimenti	0.9

2.3 SEMANTIC NETWORK

Dati un insieme di concetti in qualche modo correlati tra loro, nasce la necessità di definire una Semantic Network - Rete Semantica, per poter effettuare una analisi semantica soddisfacente.

Definizione 2.3.1: Semantic Network

Dati un insieme di n concetti $C_1, \dots C_n$ si definisce Semantic Network il grafo pesato i cui nodi rappresentano i concetti C_i e gli archi orientati le relazioni tra essi.

Più precisamente, considerati due concetti C_1 e C_2 tali che esistano due archi diretti rispettivamente da C_1 a C_2 e da C_2 a C_1:

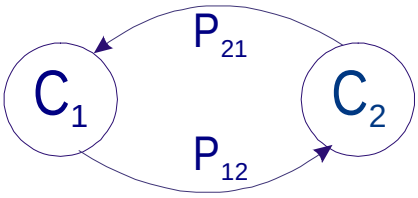

Figura 2.1: Legame tra due concetti

- P_{12} è la probabilità che il concetto C2 possa soddisfare una richiesta del concetto C1;
- P_{21} è la probabilità che il concetto C1 possa soddisfare una richiesta del concetto C2

Definiti i pesi sugli archi, possiamo dare, con riferimento alla figura 2.2, la definizione di distanza probabilistica tra due concetti.

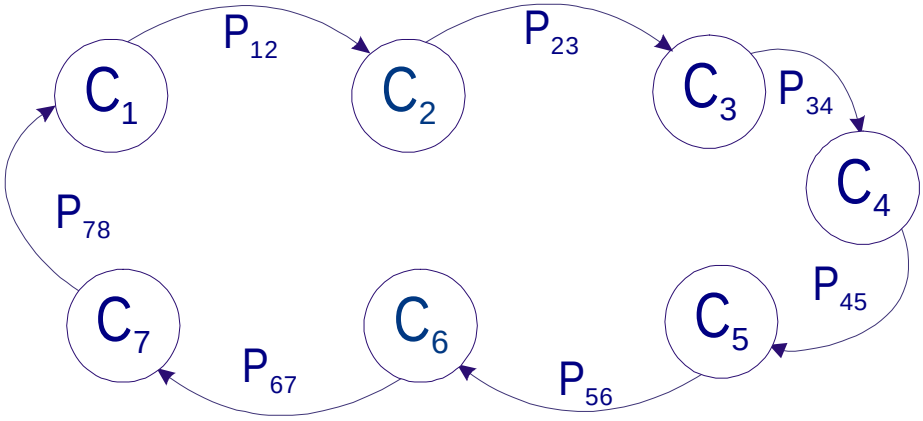

Figura 2.2: Una semplice rete semantica

Definizione 2.3.2: Distanza probabilistica

Dato un insieme di n concetti C_1, ... C_n e considerati due di tali concetti C_i e C_j, collegati da uno o più percorsi, si definisce distanza probabilistica tra i *concetti C_i e C_j la quantità*

$$d(C_i, C_j) = \max_{k \in 1..n} \left(\prod_{t=1}^{na} P_{k,t} \right)$$

dove n è il numero di percorsi esistenti tra i concetti C_i e C_j, na è il numero di archi del k-simo percorso e $P_{k,t}$ è il peso associato al t-simo arco del k-simo percorso. Si noti che $d(C_i,C_j)$ può ancora essere interpretata come la probabilità che il concetto C_j possa soddisfare una richiesta del concetto C_i e coincide con P_{ij} se esiste un arco diretto da C_i a C_j e non esistono tra gli stessi concetti percorsi migliori di quello diretto. Due concetti sono tanto più vicini dal punto di vista semantico quanto più la loro distanza probabilistica è prossima ad 1.

Esempio 2.3.1

Con riferimento alla figura 2.3, che rappresenta una possibile rete semantica specializzata sul dominio musicale, alcune delle distanze probabilistiche individuabili tra i concetti sono:

- d(genere musicale, cantante) = 0.8
- d(cantante, genere musicale) = 0.7
- d(cantautore, compositore) = 0.7·0.45 = 0.315
- d(tempi musicali, musicista) = 0.7·0.45·0.5 = 0.1575
- d(musicista, tempi musicali) = 0.6·0.45·0.6 = 0.162

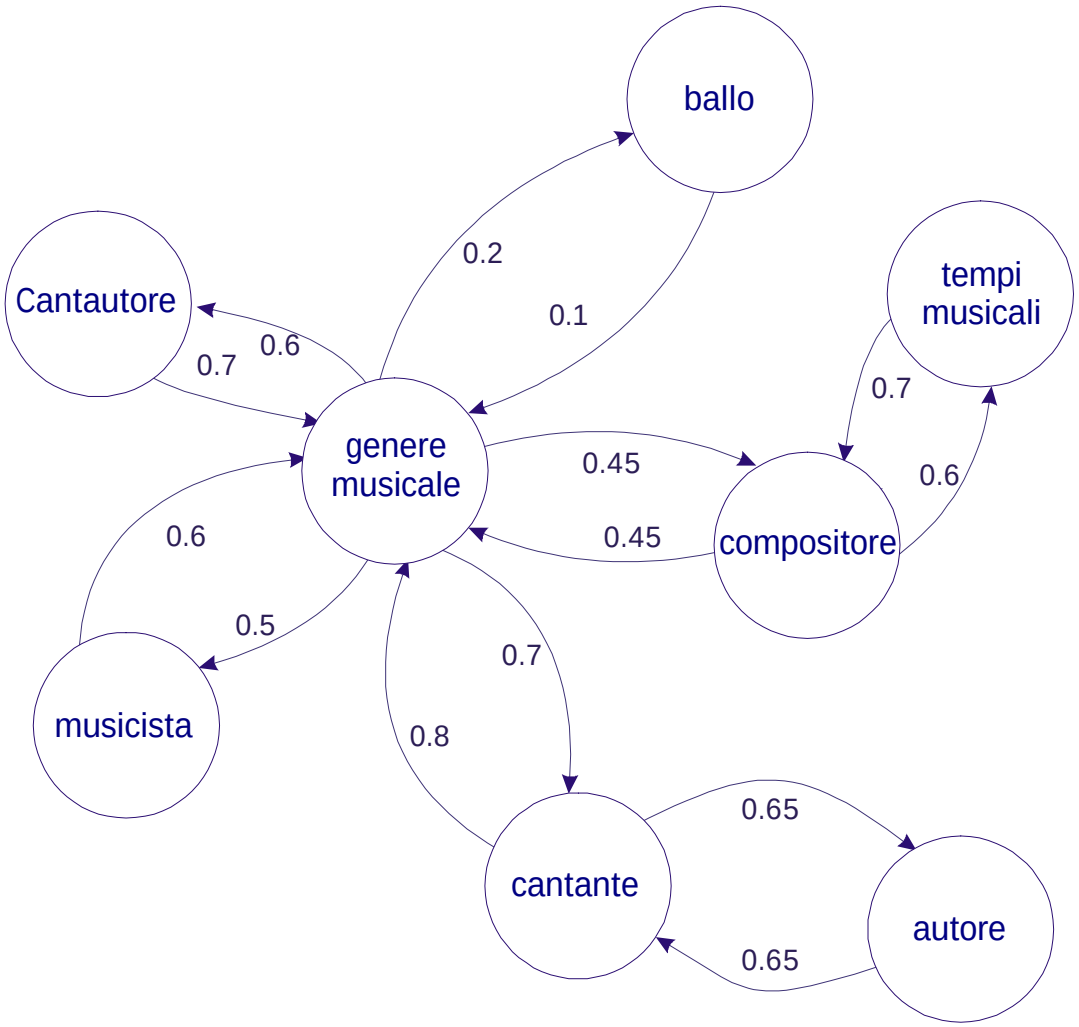

Figura 2.3: Rete semantica per il dominio musica

Facciamo ora alcune considerazioni per giustificare la distanza introdotta.

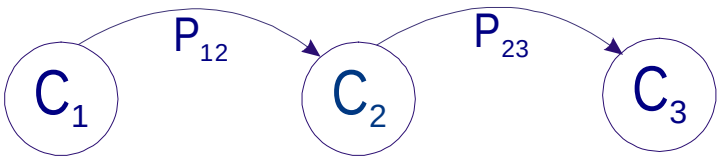

Figura 2.4: Distanza probabilistica lungo un percorso

Motiviamo prima di tutto la presenza del simbolo di produttoria. Con riferimento al semplice esempio di figura 2.4, la probabilità che il concetto C_3 possa soddisfare una richiesta del concetto C_1 è pari alla probabilità dell'evento "*C_3 soddisfa una richiesta del concetto C_2 e C_2 soddisfa una richiesta del concetto C_1*", ovvero alla probabilità

dell'intersezione di due eventi statisticamente indipendenti. Com'è noto dalla teoria della probabilità, dati due eventi statisticamente indipendenti A e B risulta:

$$P(A \cdot B) = P(A) \cdot P(B)$$

Inoltre bisogna chiarire perché si assume come distanza tra due concetti la probabilità associata al percorso probabilisticamente più corto, ossia il percorso cui è associata la distanza probabilistica più prossima ad 1. Se tra due concetti esiste più di un percorso, il percorso probabilisticamente più corto è chiaramente quello che esprime le relazioni più esplicite tra i concetti in questione e risulta dunque naturale assumere la probabilità ad esso associata come misura di similitudine tra gli stessi.

Esempio 2.3.2

Con riferimento alla figura 2.5, che rappresenta i legami tra alcuni concetti del dominio automobilistico, si ha che

- d(macchina, Honda) = 0.8·0.5 = 0.4
- d(Honda, macchina) = 0.2

Si noti come nel primo caso il percorso diretto macchina-Honda non sia quello probabilisticamente più corto. Nel secondo caso accade, invece, l'esatto contrario.

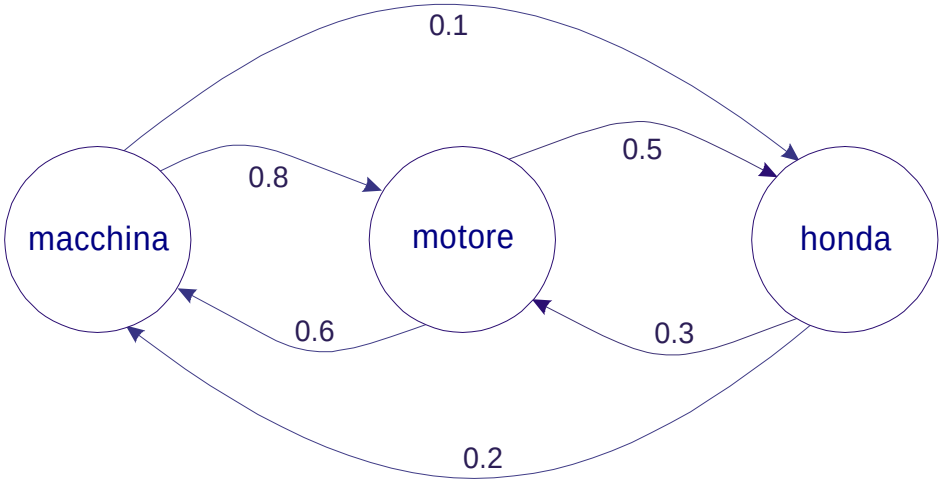

Figura 2.5: Legami tra alcuni concetti del dominio automobilistico

Per concludere, osserviamo che la distanza probabilistica soddisfa le seguenti proprietà:

- $d(C_i,C_j) \neq d(Cj,Ci)$
- $d(C_i,C_i) = 1$ $\quad \forall\, C_i \in$ insieme dei concetti
- $d(C_i,C_j) \neq d(C_i,C_k)\cdot d(C_k,C_j)$ $\quad \forall\, C_i,C_j,C_k \in$ insieme dei concetti
- $d(C_i,C_j) = 0$ $\quad \Leftrightarrow$ non esiste percorso tra C_i e C_j

2.4 LA METRICA

Fissato un dominio, per poter discriminare le pagine di interesse da quelle che non lo sono, occorre definire una metrica che consenta al sistema di dare un voto alle pagine a seconda del loro contenuto sintattico e semantico.

La metrica qui proposta prende in considerazione tre informazioni che fanno riferimento alla pagina a cui vogliamo dare un giudizio. La prima informazione è di tipo sintattico, la terza di tipo semantico e la seconda di tipo misto:

- posizione mediata della pagina;
- peso delle parole nella pagina;
- premio dell'accoppiata di vocaboli che si riferiscono a concetti correlati tra loro, pesato con la distanza dei vocaboli stessi.

POSIZIONE MEDIATA E GRADO SINTATTICO

Sia $\prod$ l'insieme delle pagine WEB. Dati n motori selezionati per la ricerca, viene assegnato ad ogni motore un peso relativo w_i, compreso tra 0 e 1, tale che:

$$\sum_{i=1}^{n} w_i = 1$$

Ciò significa che l'elenco dei link restituiti è considerato più o meno attendibile in funzione del peso assegnato al motore che ha effettuato la ricerca.

Definizione 2.4.1: Funzione posizione

Definiamo funzione di posizione

$$p_i : \prod \longrightarrow N$$

come quella funzione intera che ad ogni pagina WEB x appartenenti all'insieme $\prod$ *associa la sua posizione all'interno dell'elenco dei risultati del motore di ricerca i se la pagina viene restituita dal motore stesso, 0 nel caso in cui la pagina non fosse trovata dal motore.*

Figura 2.6: Risultato di una ricerca eseguita con Google

Esempio 2.4.1:

Considerato l'esempio di figura 2.6, si ha che la posizione della pagina www.spagnacontemporanea.it/ ha una posizione pari a 3, rispetto al motore di ricerca Google.

Definizione 2.4.2: Posizione mediata

Data una pagina WEB x, appartenente all'insieme $\prod$*, siano A l'insieme dei numeri naturali i tali che il motore di ricerca i-esimo restituisca la pagina x, e B l'insieme dei numeri naturali j tali che il motore di ricerca j-esimo non restituisca la pagina x, sia inoltre a=card(A). Si definisce posizione mediata di x, e la si indica con AP(x), la quantità:*

$$AP(x) = \frac{\sum_{i \in A} p_i(x) \cdot (w_i + \frac{\sum_{j \in B} w_j}{a})}{\sum_{i \in A} w_i}$$

Nel migliore dei casi (ogni motore di ricerca restituisce la pagina in prima posizione) si ha che AP(x)=1; il valore tende ad aumentare in funzione della maggiore posizione

della pagina all'interno del motore di ricerca e in funzione al minor numero dei motori che hanno restituito la pagina p(x). Si noti che se tutti i motori di ricerca restituissero la pagina x, la posizione mediata sarebbe la media pesata delle posizioni nei singoli motori, con i pesi attribuiti ai motori di ricerca.

Definizione 2.4.3: Grado Sintattico, SyG

Il grado sintattico di una pagina è definito come:

$$SyG\ (x) = \frac{1}{AP\ (x)}$$

Nel caso migliore il grado sintattico assume valore unitario. L'SyG è strettamente legato alla ricerca effettuata dai motori e all'ordine in cui vengono restituite le pagine; è una misura approssimativa della sola presenza nelle pagine dell'informazione ricercata. E' approssimativa poiché l'ordine delle pagine nei singoli motori è basato sempre più su motivi commerciali e sponsorizzazioni, per cui, una pagina che riguarda marginalmente l'informazione ricercata potrebbe assumere un SyG più alto rispetto ad una pagina che ne tratti approfonditamente.

CONTRIBUTO SEMANTICO

Il contributo semantico, che possiamo indicare con V, è somma di due quantità:

- Peso dei vocaboli appartenenti al dominio scelto;
- Contributo funzione delle distanze probabilistiche dei concetti trovati.

Nelle seguenti definizioni faremo prima riferimento ad un testo generico, e, successivamente, considereremo le formule nel contesto di una pagina web. Questo sarà necessario poiché la pagina web, ai fini dell'analisi, e composta da quattro parti, ognuna con un peso diverso: titolo, descrizione, parole chiavi, corpo.

Definizione 2.4.4: Grado Sintattico Semantico, SSG

Dato un dominio D e un testo x sia NT il numero dei vocaboli presenti all'interno del testo ed N il numero dei vocaboli presenti in x ed associati al dominio D, sia inoltre w_i il peso del vocabolo nel dominio D, si definisce Grado Sintattico Semantico (SSG) il valore

$$SSG\ (x) = \frac{\sum_{i=1}^{N} v_i}{NT}$$

L'SSG varia nell'intervallo [0,1], nel migliore dei casi il valore è unitario.

Il peso assume un significato sintattico poiché consideriamo i vocaboli in modo isolato e un significato semantico dovuto alla capacità del singolo vocabolo (data da ω_i) di rappresentare un dato dominio.

Ora, poiché ad ogni vocabolo è associato al più un concetto in un dato dominio, possiamo parlare anche di concetti appartenenti al dominio D trovati in un testo.

Definizione 2.4.4: Distanza tra due concetti di un dominio D

Dato un testo x e al suo interno una coppia di concetti C_i,C_j appartenenti al dominio D si definisce distanza tra i concetti, e la si indica con DIST(C_i,C_j) la differenza tra le posizioni dei concetti nel testo:

$$DIST(Ci,Cj) = |Posizione(C_i) - Posizione(C_j)|$$

Definizione 2.4.4: Distanza probabilistica normalizzata

Dato un testo x e al suo interno una coppia di concetti C_i,C_j appartenenti al dominio D si definisce distanza probabilistica normalizzata, e la si indica con NPD(Ci,Cj), la distanza probabilistica che i due concetti hanno nel dominio D diviso la distanza tra i due concetti nel testo:

$$NPD(C_i, C_j) = \frac{d(C_i, C_j)}{DIST(C_i, C_j)}$$

Definizione 2.4.5: Grado Semantico SeG

Dato un testo x e al suo interno la sequenza dei concetti di un dominio D espressi dai relativi vocaboli, sia NC il numero totale dei concetti, si definisce grado semantico la somma delle distanze probabilistiche normalizzate di ciascun concetto con tutti i successivi:

$$SeG(x) = \sum_{h=1}^{NC} \sum_{k=h+1}^{NC} NPD(C_i, C_j)$$

Il grado semantico può assumere anche un valore molto elevato, in funzione del numero dei concetti trovati e del valore di NPD. Supposto che tutti i concetti abbiano una distanza probabilistica normalizzata unitaria si ha che:

$$SeG(x)_{\max} = \frac{NC^2 + NC}{2}$$

Definizione 2.4.5: Grado Semantico Normalizzato, NseG

Il Grado Semantico Normalizzato è definito come:

$$NSeG(x) = \sum_{h=1}^{NC} \sum_{k=h+1}^{NC} \frac{2 \cdot NPD(C_i, C_j)}{NC^2 + NC}$$

Per applicare le formule esposte ad una pagina web è necessario considerare separatamente il testo del titolo, della descrizione, delle parole chiavi e del corpo. Per ogni parte si dovrà calcolare l' SSG e l' NSeG considerando, per le parole chiavi, la distanza fra concetti con valore unitario (le parole chiavi rappresentano un elenco di vocaboli).

Definizione 2.4.6: Grado Sintattico-Semantico di una pagina Web

Il Grado Sintattico-Semantico di una pagina WEb è dato da:

$$SSG(x) = \sum_{i \in \{t,d,k,c\}}^{NC} r_i \cdot SSG_i(x)$$

dove r_i indica il peso assegnato al titolo, descrizione, parole chiavi e corpo e deve essere tale che:

$$r_t + r_d + r_k + r_c = 1$$

Definizione 2.4.7: Grado Semantico di una pagina Web

Il Grado Semantico di una pagina WEb è dato da:

$$NSeG(x) = \sum_{i \in \{t,d,k,c\}}^{NC} r_i \cdot NSeG_i(x)$$

dove r_i indica il peso assegnato al titolo, descrizione, parole chiavi e corpo e deve essere tale che:

$$r_t + r_d + r_k + r_c = 1$$

Nel calcolare il grado semantico del testo associato al corpo della pagina web, concetti molto distanti fra loro contribuiscono in minima parte al valore del $NSeG_c(x)$, per cui è possibile non tenerne affatto conto e introdurre quindi

Definizione 2.4.8: Grado Semantico Normalizzato Approssimato

Sia fissata una soglia T (threshold), il Grado Semantico Normalizzato Approssimato è definito come:

$$\overline{NSeG}_c(x) = \sum_{h=1}^{NC} \sum_{k=h+1}^{NC} f\left(\frac{2 \cdot NPD(C_i, C_j)}{NC^2 + NC}, T\right)$$

$$f(x,T) = x \Leftrightarrow x \geq T \qquad f(x,T) = 0 \Leftrightarrow x < T$$

VOTO COMPLESSIVO

Il voto complessivo della pagina web, GG(x) è calcolato tenendo conto dei tre diversi contributi:

- Grado Sintattico, SyG (def. 2.4.3), con peso pari a k_y;
- Grado Sintattico-Semantico, SSG (def. 2.4.6), con peso pari a k_{ss};
- Grado Semantico Normalizzato, NSeG (def. 2.4.7-2.4.8), , con peso pari a k_e.

per cui si ha:

$$GG(x) = \frac{k_y SyG(x) + k_{ss} SSG(x) + k_e \overline{NSeG}(x)}{k_y + k_{ss} + k_e}$$

Orientativamente, il peso maggiore sarà assunto dal NSeG, mentre quello minimo dal SyG.

Capitolo 3

Il sistema

3.1 Architettura
3.2 Sistema distribuito
3.3 Base Dati del Sistema

E' possibile definire un'architettura ad alto livello del sistema che preveda la necessità di sottoporre la query ai motori di ricerca selezionati dall'utente. In questa fase, il sistema dovrà dialogare con ciascun motore di ricerca al fine di ottenere un'insieme di link che verrà archiviato, insieme alle relative pagine web, nel Web Repository. In una seconda fase, le pagine archiviate verranno preprocessate al fine di eliminare tutto ciò che non deve essere analizzato da un punto di vista semantico e isolarne le parti essenziali: titolo, descrizione, parole chiavi e corpo. Successivamente, il Miner effettuerà l'analisi semantica, al fine di assegnare un punteggio ad ogni documento.

Tale funzionamento non garantisce una ricerca efficiente. Vi è una fase iniziale, legata al download delle pagine web, che è caratterizzata da lunghi periodi di inattività del sistema. E' necessario, fin da adesso, prevedere uno sviluppo distribuito del sistema in moduli separati in grado di collaborare e autosincronizzarsi.

Le informazioni relative all'intero sistema verranno inserite in una base dati che sarà presentata in questo capitolo, a livello progettuale, utilizzando il modello Entità-Associazione.

3.2 ARCHITETTURA

L'architettura del nostro sistema è basata su:

- Search Engine Wrapper:

  sottopone le interrogazioni ai motori di ricerca;

- Web Spider:

  individua le pagine web e le salva nel DB;

- Document Preprocessor:

  prepara le pagine per l'analisi semantica;

- Miner:

  effettua l'analisi semantica.

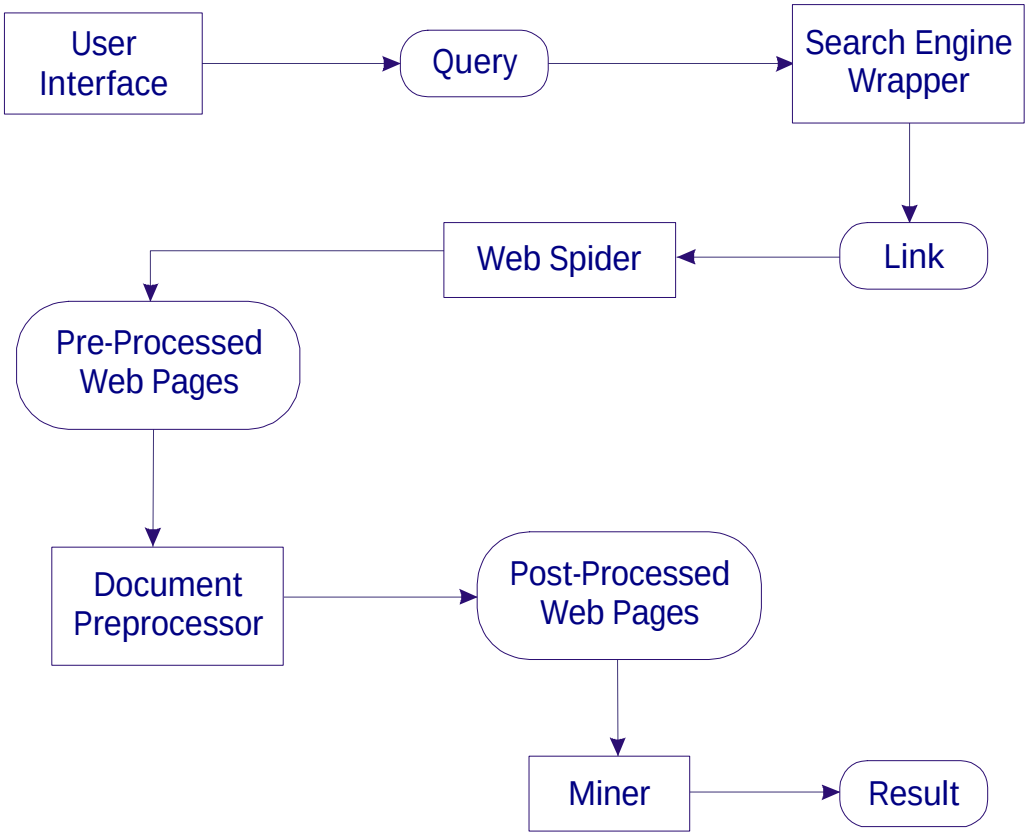

Foto 3.1: Architettura ad alto livello del sistema

SEARCH ENGINE WRAPPER

Il modulo Serch Engine Wrapper prende la query inserita mediante l'interfaccia utente e la adattata alle specifiche sintassi dei motori di ricerca, mediante il Query Adapter, creando la stringa di interrogazione per i singoli motori di ricerca scelti per la ricerca.

Allo scopo di rendere del tutto trasparente per l'utente l'interrogazione dei motori di ricerca, il Search Engine Wrapper si occupa della sottomissione delle query adattate dal Query Adapter ai motori di ricerca e di prelevarne i risultati. Il Search Engine Submitter sottopone la query ad un motore di ricerca ottenendo la pagina con i risultati da esso restituiti ed il Parser analizza tale pagina allo scopo di prelevarne i link in essa contenuti.

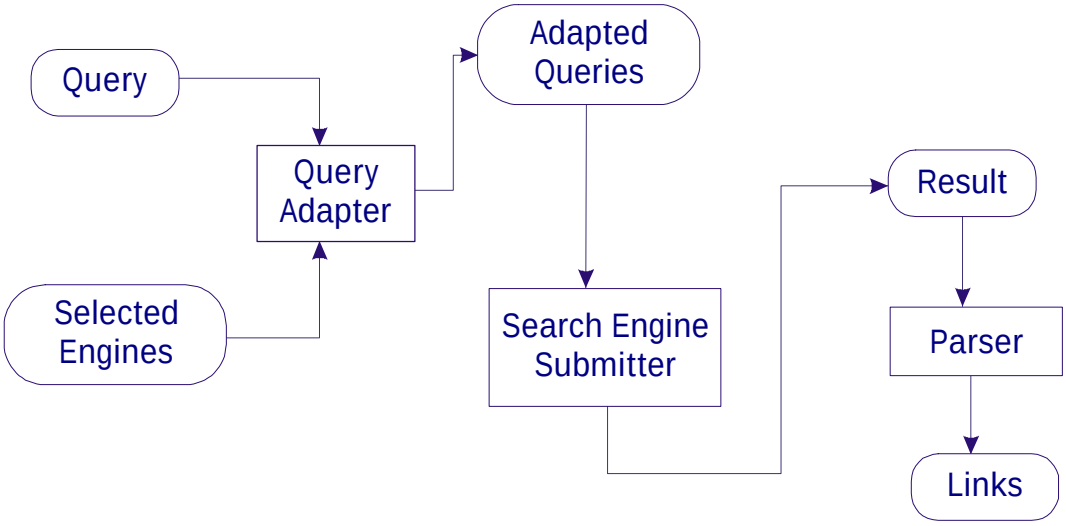

Foto 3.2: Il Search Engine Wrapper

WEB SPIDER

Il compito del Web Spider è quello di individuare le pagine corrispondenti ai link restituiti dai motori di ricerca e di memorizzarle nel repository locale. Le pagine vengono localizzate dal Web Catcher, mentre il Repository Builder provvede ad inserirle nel Database del sistema.

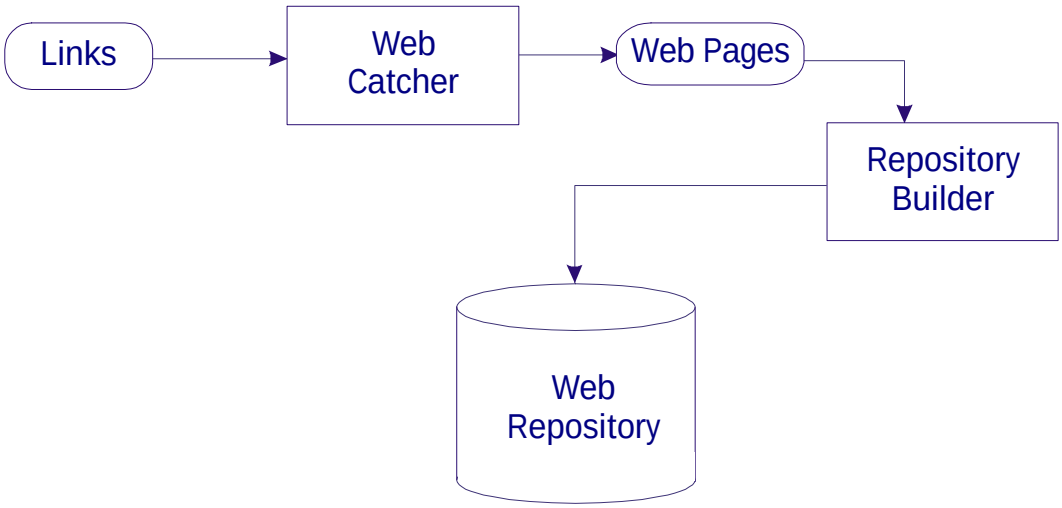

Figura 3.3: Schema del Web Spider

DOCUMENT PREPROCESSOR

Alla fine delle due precedenti elaborazioni troveremo nel Web Repository un elenco di pagine Web legate ad una data interrogazione.

Le pagine Web sono costituite da tre tipi di informazioni:

- Tag HTML

  il cui significato riguarda essenzialmente la formattazione del testo nella pagina.

- Meta Tag HTML

  in cui l'autore della pagina può inserire una descrizione sintetica del sito, le parole chiave che lo caratterizzano, l'autore del sito ed, eventualmente, altre informazioni. La presenza dei Meta Tag non è obbligatoria ed è cura dell'autore inserirli. Peraltro, l'utente è spinto ad immettere un contenuto coerente in tali campi in quanto alcuni motori di ricerca ne fanno uso per semplificare l'indicizzazione delle pagine e, talvolta, premiano le pagine in cui i Meta Tag sono particolarmente significativi.

- Testo

  rappresenta l'informazione che l'autore vuole trasmettere.

Il contenuto semantico di una pagina è chiaramente concentrato nel testo, ma anche i Meta Tag hanno la loro importanza, in quanto sintetizzano il contenuto semantico della pagina. Nella metrica utilizzata dal sistema realizzato, vengono considerati i seguenti campi della pagina HTML:

- Testo;
- Titolo;
- Meta Tag “Description”;
- Meta Tag “Keywords”.

Per estrarre tali informazioni dalla pagina è necessario analizzarne i tag HTML. Tale operazione viene effettuata dal Document Preprocessor, che estrapola il contenuto semantico dalla pagina HTML, lo separa nelle quattro componenti succitate e le memorizza nel database di sistema.

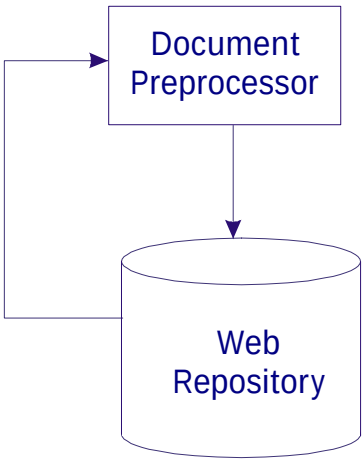

Foto 3.4: Schema del Document Preprocessor

MINER

Il cuore del Miner è costituito dalla rete semantica. Essa contiene le informazioni necessarie per analizzare il contenuto semantico di una pagina e misurarne l’attinenza con un certo dominio. A tal fine si considera la metrica esposta nel capitolo precedente, che prevede le seguenti informazioni:

- Posizione mediata della pagina (*SyG*);

- Peso dei vocaboli presenti (SSG);
- Premio dell'accoppiata di vocaboli che si riferiscono a concetti tra loro correlati, pesato con la distanza dei vocaboli (SeG);

Il primo contributo viene determinato dal Web Catcher che valuta l'insieme dei motori di ricerca che ha restituito una data pagina. Gli ultimi due vengono valutati dal Miner. A tal fine, la prima operazione che esso deve effettuare è l'eliminazione delle stop-words, ossia di quelle parole che non danno contributo alla semantica del documento. Tali parole sono tipiche di ogni lingua e, poiché nel nostro sistema viene considerata esclusivamente la lingua italiana, la lista delle stop-words della lingua italiana è memorizzata nel Database di sistema.

Ottenuto il documento depurato dalle stop-words, occorre valutare il SSG e il SeG del documento; questo avviene consultando il Database del sistema che contiene le informazioni relative alla rete semantica.

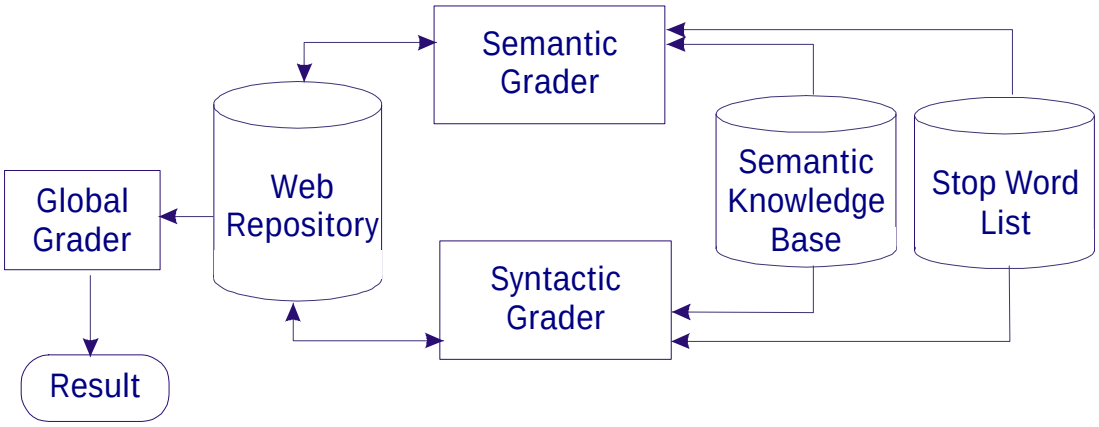

Figura 3.5: Schema Architetturale del Miner

L'architettura del miner è molto flessibile: sarà sempre possibile cambiare o modificare la metrica: basta agire sulla Semantic Knowledge Base.

3.3 SISTEMA DISTRIBUITO

Il sistema deve essere progettato in modo che più utenti possano utilizzarlo contemporaneamente: deve essere sviluppato secondo il modello client-server.

Da quanto esposto nel paragrafo precedente, si evince che, la ricerca semantica è abbastanza complessa e richiede l'interazione con elementi che non sono sotto il diretto controllo del sistema.

La richiesta di un set di pagine ad un determinato motore richiede un tempo che è collegato al tipo di interrogazione da effettuare, al numero di link da prelevare e al numero di utenti che sono contemporaneamente connessi con il motore; l'elenco delle pagine web viene determinato anche in funzione al numero di motori scelti.

Il tempo per la ricerca delle pagine web e il relativo download nel Web Repository è direttamente legato ai tempi di accesso alle pagine web e, più in generale, al traffico in rete.

Inoltre, l'operazione di preprocessing e quella di mining vengono eseguite in successione su tutti i documenti, a valle di quella di spidering.

In base a tali considerazioni si evince che, l'efficienza potrebbe aumentata notevolmente se:

- l'operazione di spider avesse un veloce collegamento ad internet;
- se i documenti potessero essere preprocessati appena inseriti nel repository;
- se il mining potesse essere attivato su ogni documento appena processato.

Tutto ciò implica un'elaborazione concorrente in cui il sistema non deve avere periodi di inattività.

Inoltre, le tre operazioni fondamentali hanno esigenze hardware diverse:

- lo Spider esigenze legate al collegamento internet;
- il Preprocessor e il Miner richiedono una maggiore memoria e velocità di elaborazione, oltre che ad un uso intensivo della base dati.

Si osserva poi, che, le tre elaborazioni (spidering, preprocessing, mining), sostanzialmente, hanno vita autonoma e, se le consideriamo in maniera concorrente, si sincronizzano solo nella fine delle proprie elaborazioni.

Considerando tutto ciò, si è pensato di sviluppare il sistema su tre server diversi, ciascuno con una funzionalità ben precisa e che siano in grado di sincronizzarsi sulle attività da svolgere:

- Server Spider

  comprende il Search Engine Wrapper e il Web Spider

- Server PreProcessor

  comprende il Document Preprocessor

- Server Miner

  comprende il miner

La richiesta al sistema sarà effettuata dal client che invierà un messaggio ad ogni server per iniziare la propria elaborazione.

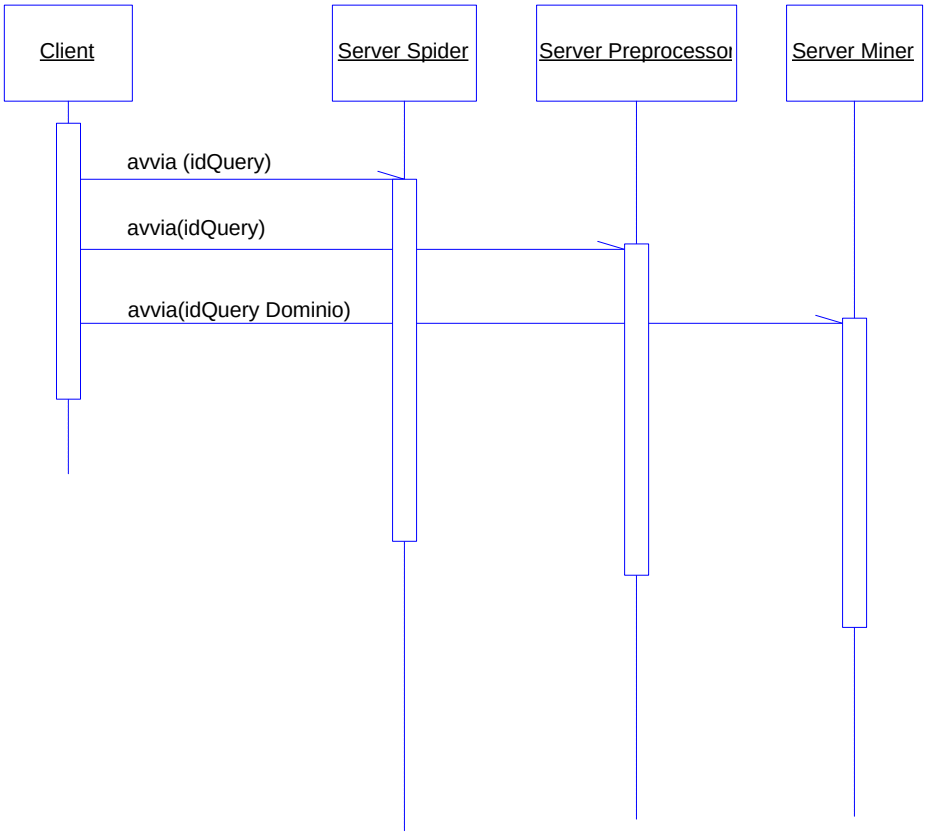

Figura 3.6a: diagramma di sequenza tra Client e Server

Poiché l'utilizzo è tramite Web e poiché i tempi di elaborazione, per quanto piccoli, non sono trascurabili, il client invia dei messaggi non bloccanti ai server, in modo da liberare subito l'utente dalla connessione al sistema.

L'utente, dopo qualche secondo, o successivamente, potrà prendere visione dei risultati interrogando direttamente la Base Dati.

Consideriamo la figura 3.6a, si ha:

1. La query è stata precedentemente inserita nel DB del sistema;

2. Il client avvia lo spidering con il messaggio avvia(idQuery) rivolto al server Spider, passandogli il riferimento della Query nel DB del sistema. In tal modo lo spider inizia a interrogare i motori di ricerca selezionati e a scaricare le pagine relative ai documenti individuati. La sua attività termina quando:
    a. non ci sono più pagine da scaricare.

3. Il client avvia il preprocessing con il messaggio avvia(idQuery) rivolto al server Preprocessor, passandogli il riferimento della Query nel DB del sistema. Il Preprocessor interroga la base dati e quando trova nuove pagine le elabora. La sua attività termina quando:
    a. lo spider ha terminato;
    b. non ci sono più pagine da elaborare.

4. Il client avvia il mining con il messaggio avvia(idQuery, Dominio) rivolto al server Miner, passandogli il riferimento della Query nel DB del sistema e il dominio scelto dall'utente. Il Miner interroga la base dati e quando trova nuovi documenti già preprocessati, ne effettua il mining. La sua attività termina quando:
    a. il Preprocessor ha terminato di elaborare i documenti;
    b. non ci sono più documenti da elaborare .

I tre server si sincronizzeranno nella base dati del sistema, indicando lo stato della propria elaborazione, ovvero se è in corso o se è terminata.

Strutturare il sistema su tre server separati, con comunicazione reciproca minima, che tra l'altro avviene attraverso la base dati, ci consente di progetto autonomamente i tre server osservando esclusivamente il modello di comunicazione indicato e l'obbligo

di salvare il proprio stato nella base dati del sistema. I tre server possono essere organizzati concretamente, in tre package autonomi, con dipendenza da un package comune che gestisce l'accesso alla base dati.

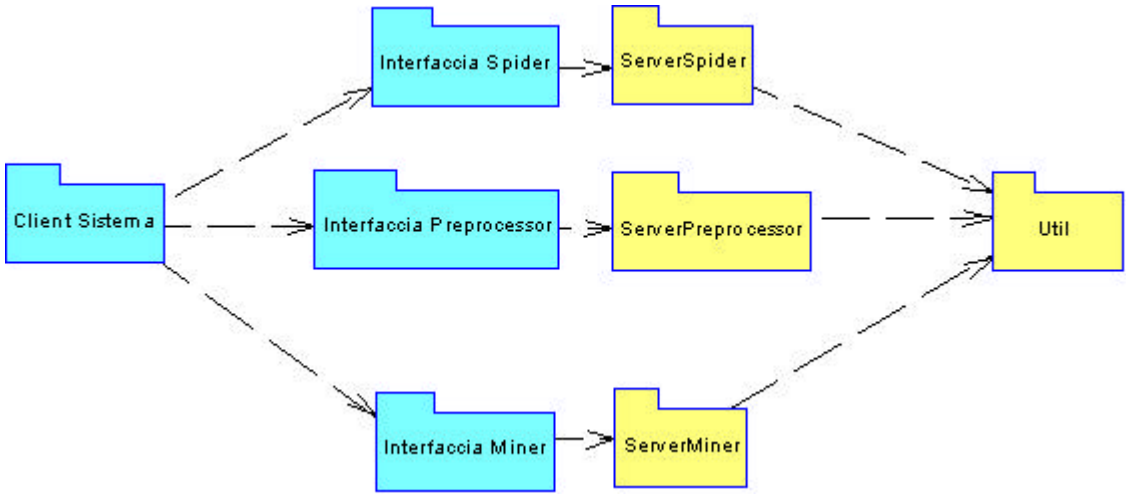

Figura 3.6b: diagramma dei package concreti (giallo) e indicativi del sistema (turchese)

Nella figura 3.6b i package relativi alle interfacce vengono indicati con un diverso colore, poiché sono indicativi: hanno lo scopo di evidenziare la necessità di rendere indipendente lo sviluppo dei server dalla propria interfaccia, che è quella indicata in figura 3.6a. Nel capitolo 6 sarà evidenziato il modo in cui vengono create e gestite le interfacce.

COMUNICAZIONE TRA GLI UTENTI E IL SISTEMA

Una delle specifiche prevede che gli utenti debbano avere accesso al sistema tramite web. Dalla figura 3.7 si evince la comunicazione tra utente e sistema per ogni nuova query:

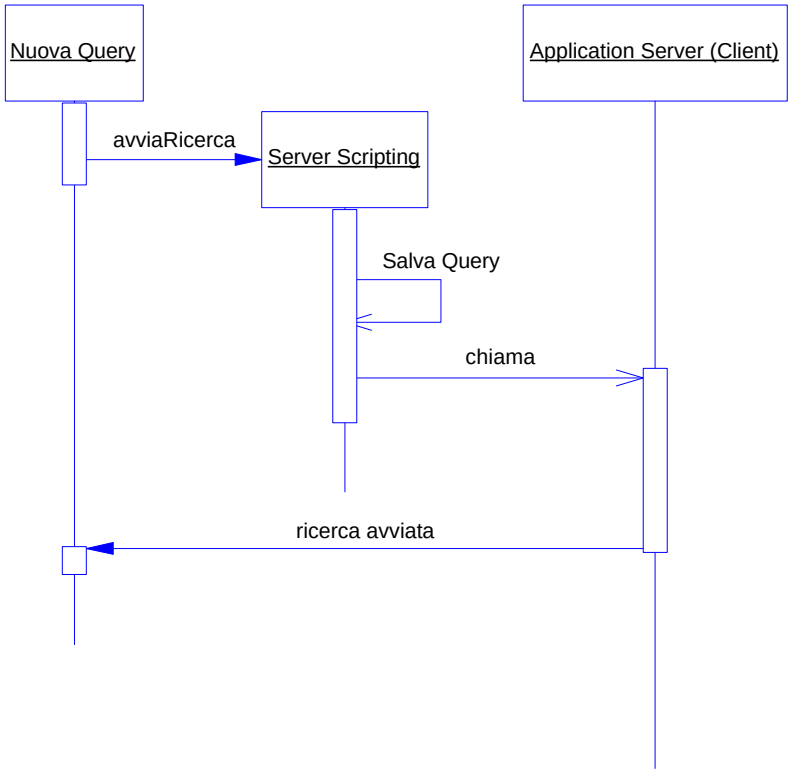

Figura 3.7: avvio di una nuova ricerca

In questo caso l'utente interagendo con una pagina web inserisce una nuova query. L'avvio della ricerca attiverà uno scripting lato server con accesso diretto alla base dati per la memorizzazione della query; successivamente viene passato il controllo al client del sistema che attiva i server e comunica all'utente che l'operazione è stata avviata.

L'utente potrà prendere visione di tutte le query inoltrate al sistema e completate, e potrà visualizzarne i risultati. In questo caso i dati verranno direttamente prelevati dalla base dati del sistema a cui gli script lato server avranno accesso diretto.

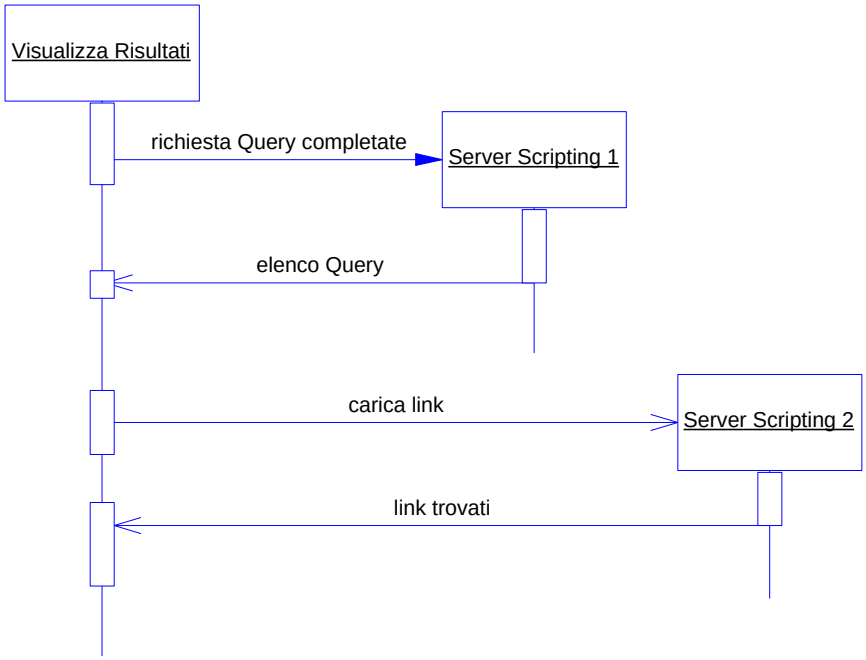

Figura 3.8: controllo dei risultati ottenuti

L'amministratore del sistema potrà intervenire sulla base dati utilizzando un insieme di pagine web con accesso protetto che attivano sempre script lato server.

L'utilizzo di script lato server con accesso diretto alla base dati permettono di non sovraccaricare il sistema e utilizzarlo esclusivamente per la ricerca semantica.

Da quanto detto è evidente che l'intera interfaccia utente del sistema deve essere realizzata tramite html dinamico.

DIAGRAMMA DEI COMPONENTI E DI DEPLOYMENT

I componenti che fanno parte del nostro sistema sono:

- ✓ Browser Web
  1. è utilizzato per avere accesso al sistema
- ✓ Web Server
  1. permette di gestire la comunicazione con il Browser Web
  2. gestisce gli script lato server che accedono alla base dati

- ✓ Application Server
    1. rappresenta l'applicazione client del sistema. Per ogni nuova query attiva la comunicazione con i server.
- ✓ Server Spider
- ✓ Server PreProcessor
- ✓ Server Miner
- ✓ Base Dati del sistema
    1. contiene il Web Repository;
    2. le informazioni relative alle query e allo stato delle elaborazioni;
    3. informazioni relative alla rete semantica.

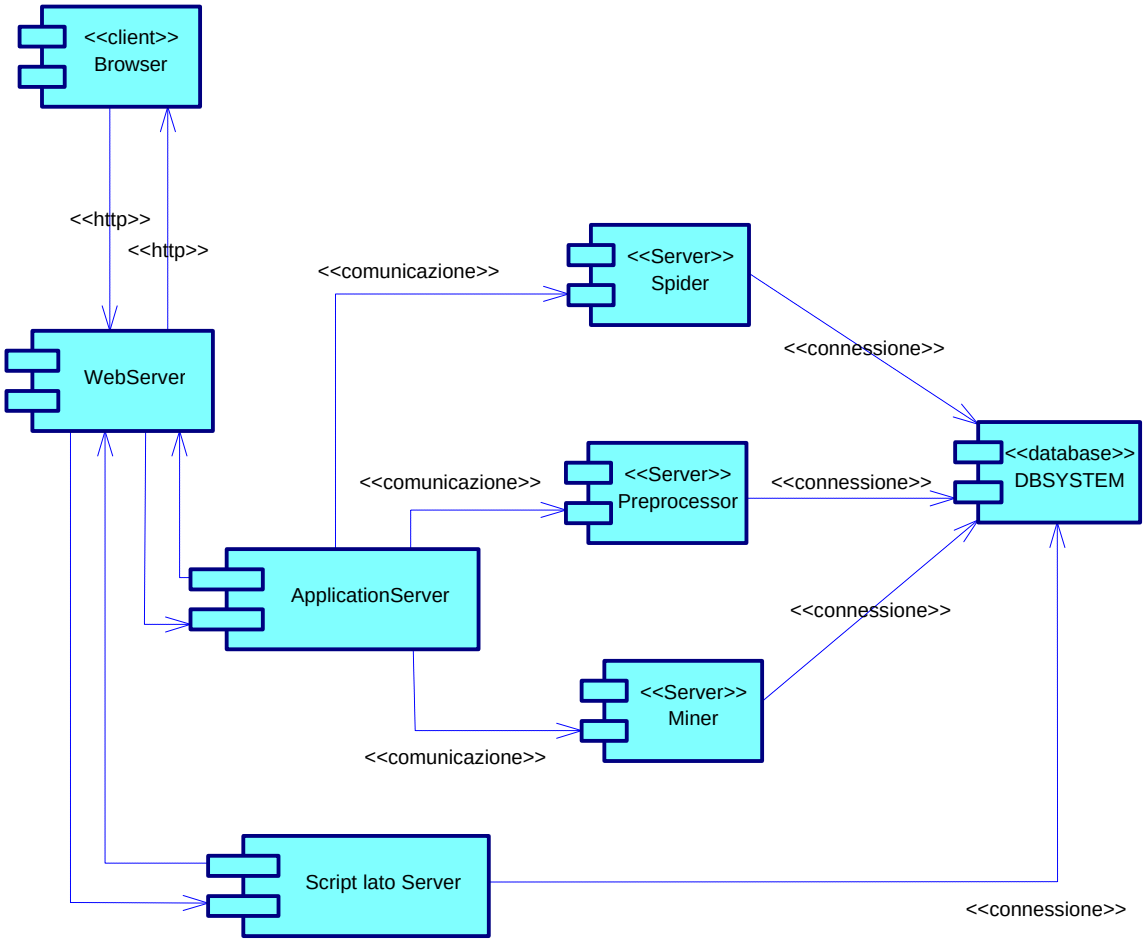

Foto 3.9: diagramma dei componenti del sistema

Per quanto riguarda la collocazione dei componenti sulle risorse hardware si può pensare che i server del sistema siano collocati su nodi distinti; il web server e le applicazioni lato server su un altro nodo. Il Database può essere presente su uno di questi nodi oppure su uno a parte: dipende dall'attività svolta dal sistema e in particolare dagli accessi alla base dati.

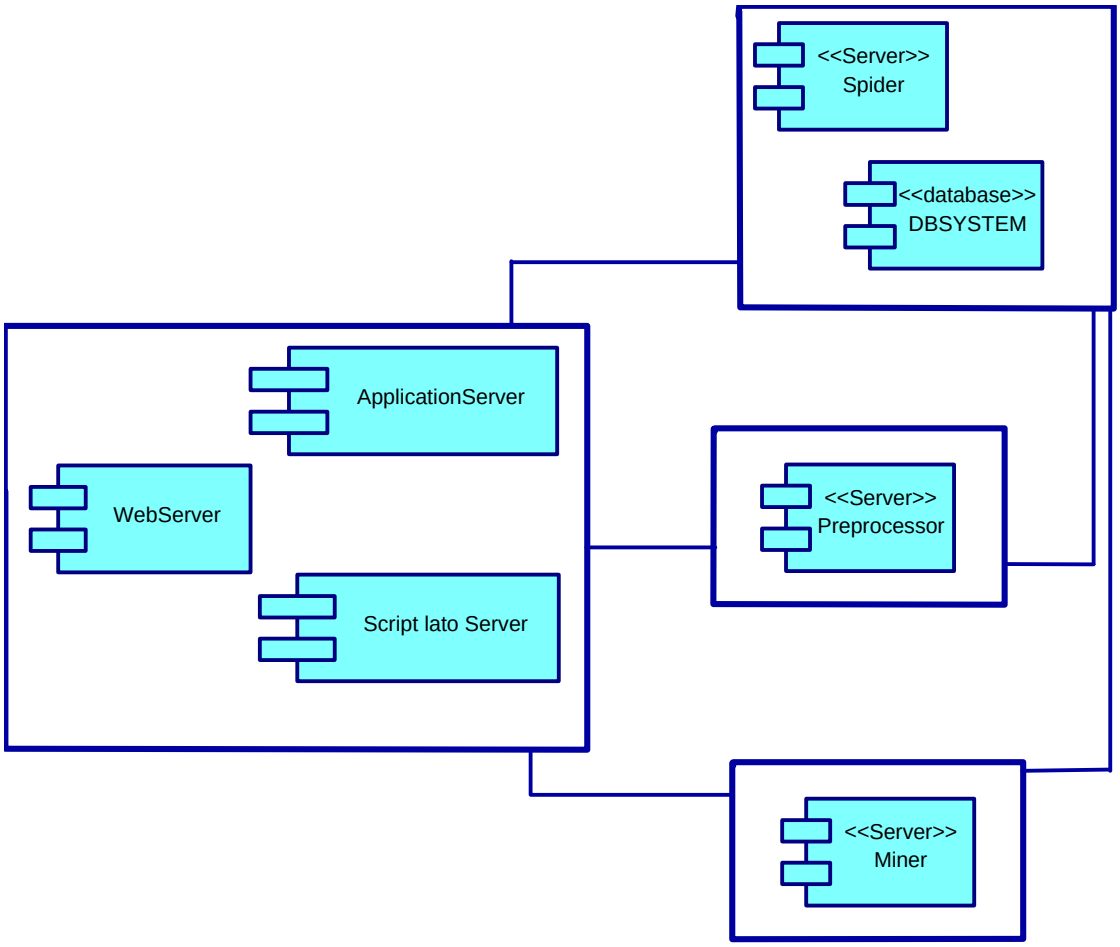

Figura 3.10: diagramma di Deployment del sistema

3.4 BASE DATI DEL SISTEMA

La base dati di supporto al sistema è formata essenzialmente dalle seguenti parti:

- dati relativi ai Motori di Ricerca che è possibile utilizzare;
- dati relativi alle caratteristiche delle query richieste al sistema;
- dati relativi allo stato di elaborazione dei server;
- Web Repository;
- Semantic Knowledge Base (SKB), ossia informazioni sull'ontologia.

Fatta eccezione per la SKB, le altre parti sono in qualche modo associate fra loro, per cui è possibile suddividere il progetto della Base Dati in due grandi parti:

- Repository;
- Semantic Knowledge Base;

Nei paragrafi seguenti presenteremo il relativo progetto Entità-Associazione per ciascuna delle due parti.

REPOSITORY

Il Repository rappresenta tutti i dati relativi ai motori, alle query e ai documenti da elaborare o elaborati.

Le entità che vi troviamo sono:

1) Query

   Le occorrenze rappresentano le query sottoposte al sistema.

2) MotoriDiRicerca

   E' l'entità associata ai motori di ricerca.

3) Link

   Ogni occorrenza rappresenta un link, inteso come universal reference locator http relativo a documenti individuati e scaricati nel sistema.

4) Documenti

Rappresenta le pagine web individuate e memorizzate con tutto il codice HTML.

5) DocumentiPreProcessati

Le occorrenze di tale entità si riferiscono a documenti già preprocessati e pronti all'elaborazione semantica.

6) Stato

Indica, per ogni query, lo stato di elaborazione dei server. Ogni occorrenza è relativa ad una query e ad un server. Se è presente indica che quel server, per quella query ha terminato l'elaborazione, altrimenti è ancora in corso.

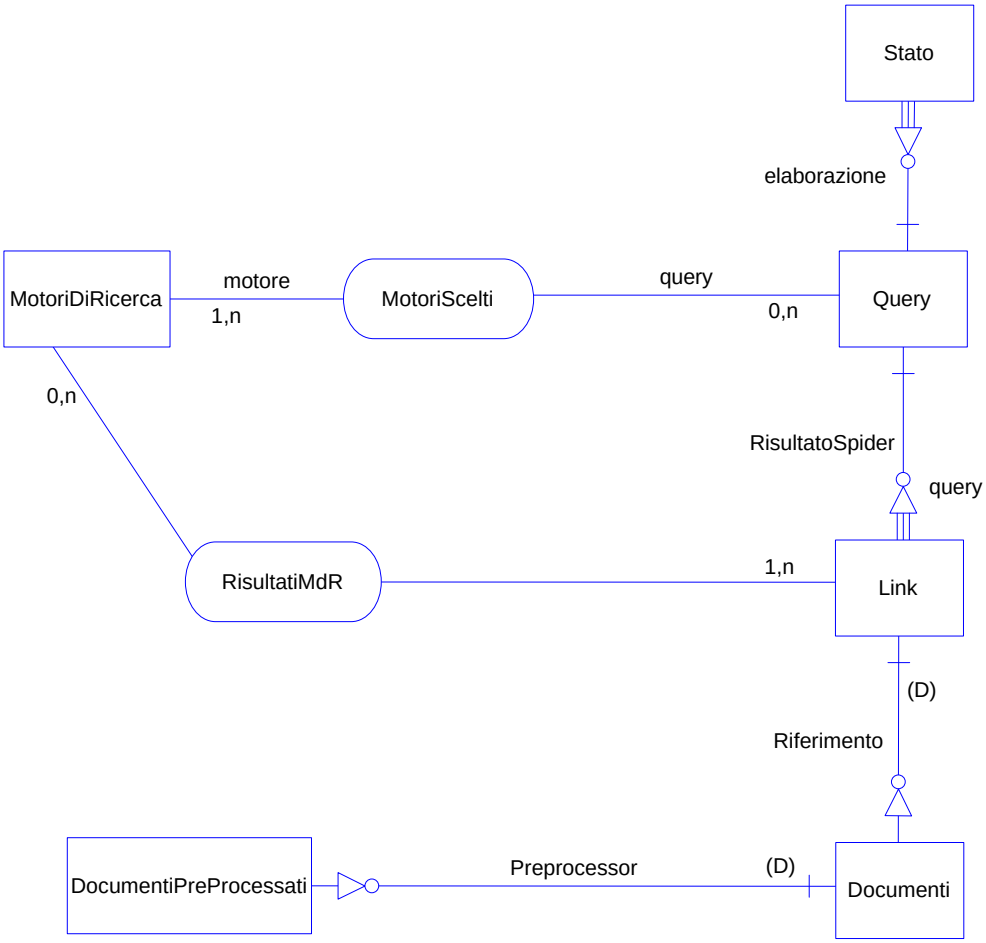

Figura 3.11a: schema E-R del repository

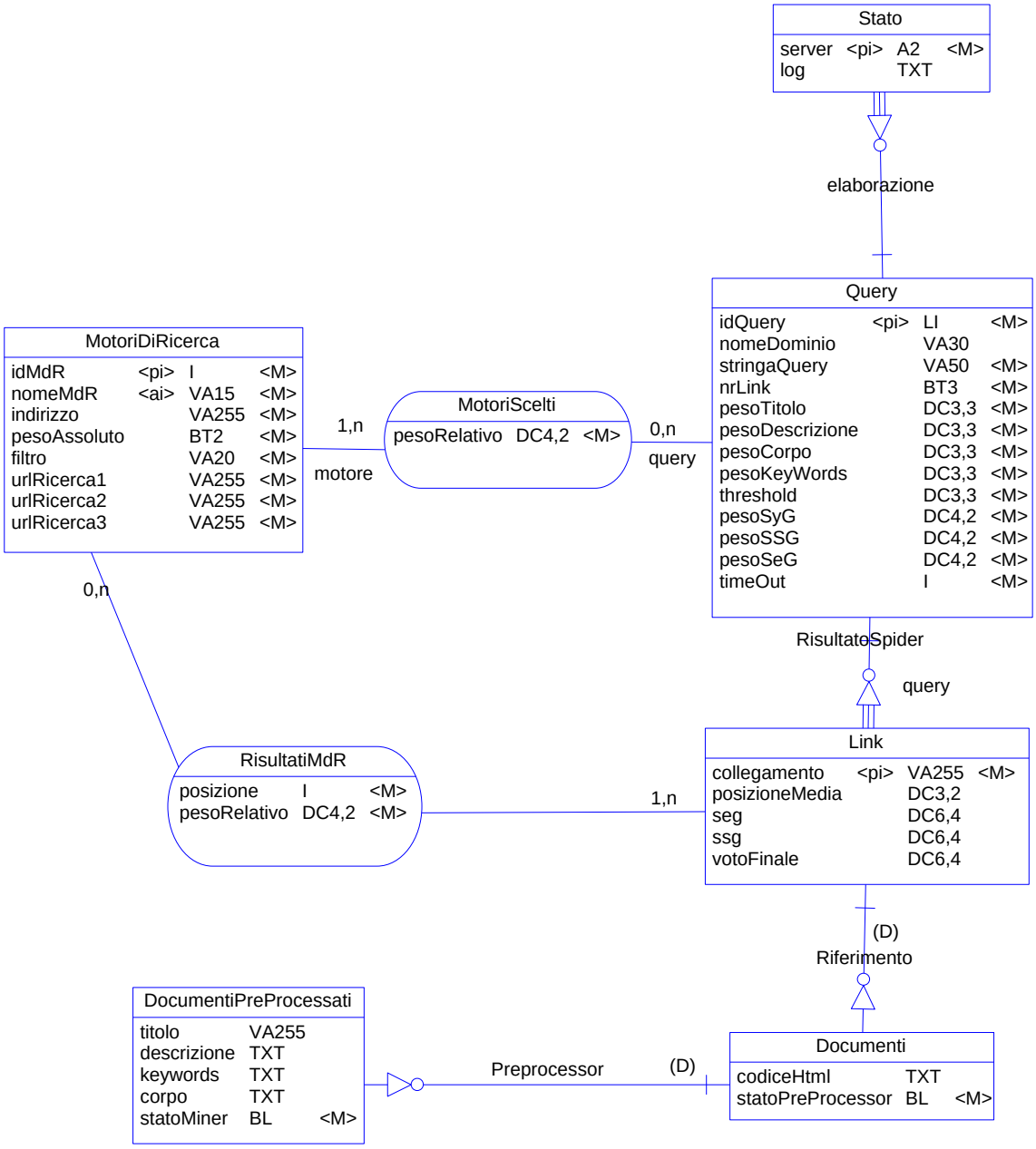

Figura 3.11b: schema E-R di dettaglio del repository

Si noti che:

- l'entità Link ha un identificatore esterno con l'entità query
- l'entità Documenti ha un identificatore esterno con l'entità Link
- l'entità DocumentiPreProcessati ha un identificatore esterno con l'entità Documenti.
- l'entità Stato ha un identificatore esterno con l'entità Query;

Inoltre le associazioni permettono di indicare l'insieme dei motori scelti per una data query (MotoriScelti, indicando per ogni associazione il peso relativo assegnato) e di rap-

presentare l'insieme dei motori che hanno individuato un determinato Link (RisultatiMdR, indicando per ogni associazione la posizione assunta dal link nel motore di ricerca e il peso relativo del motore).

Di seguito indichiamo alcuni dettagli sulle entità del Repository.

MotoriDiRicerca

MotoriDiRicerca			
idMdR	<pi>	I	<M>
nomeMdR	<ai>	VA15	<M>
indirizzo		VA255	<M>
pesoAssoluto		BT2	<M>
filtro		VA20	<M>
urlRicerca1		VA255	<M>
urlRicerca2		VA255	<M>
urlRicerca3		VA255	<M>

Figura 3.12a: MotoriDiRicerca, dettaglio degli attributi

I primi tre attributi rappresentano l'identificativo primario (idMdR), il nome (nomeMdR) e l'indirizzo http (indirizzo) del motore di ricerca.

Vi è poi, il pesoAssoluto, un valore numerico da 1 a 10 che rappresenta l'efficacia che l'amministratore assegna al motore di ricerca. Tale informazione viene utilizzata per consigliare all'utente il peso relativo da assegnare.

Un altro attributo molto importante è il filtro, che consente di filtrare dalla pagina dei risultati i link che non sono attinenti alla ricerca. Di solito esso coincide con il nome del motore di ricerca (per Google sarà google).

Ogni motore di ricerca, tra l'altro, si contraddistingue per il modo in cui rappresenta la congiunzione tra vocaboli e per il modo in cui gestisce l'elenco totale dei link trovati, ovvero il numero di link per pagina.

Per rendere la gestione dei motori di ricerca il più semplice possibile, è necessario per ogni motore inserire tre url-http:

- urlRicerca1
- urlRicerca2
- urlRicerca3

Ognuno di questi attributi si riferisce alla ricerca della frase zucchero zucchero, nel relativo motore di ricerca, e all'elenco della prima, seconda e terza pagina dei risultati.

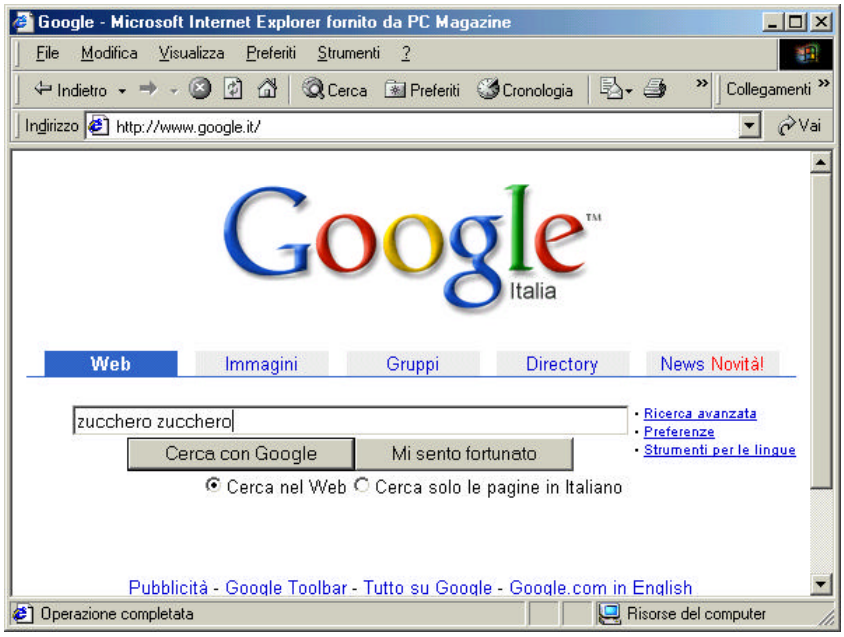

Figura 3.12b: inserimento della stringa zucchero zucchero nel motore google

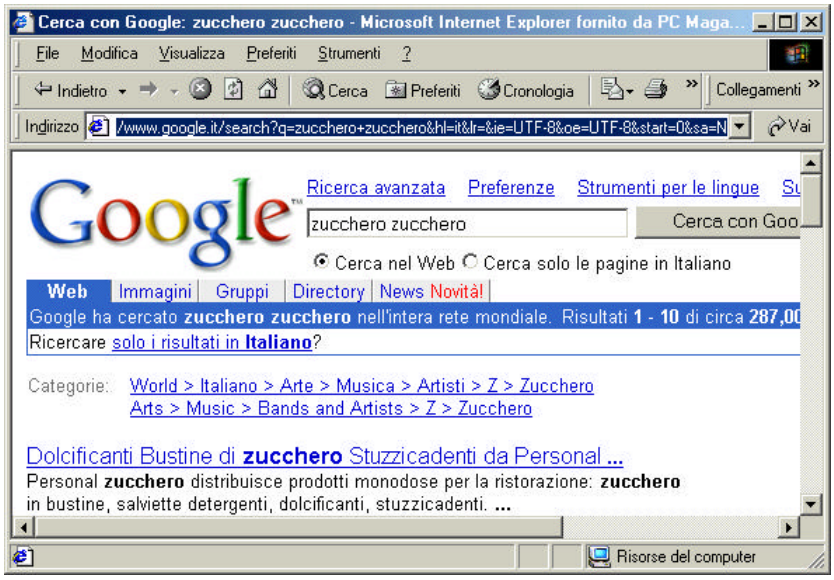

Figura 3.12c: selezione dell'indirizzo per la prima pagina dei risultati

Dalla figura 3.12c si osserva che è necessario copiare l'indirizzo. Tale operazione deve essere eseguita per le prime tre pagine dei risultati, precisamente bisognerà copiare prima l'indirizzo relativo alla terza pagina, poi quello della seconda ed infine quello della prima. Ciò è necessario perché alcuni motori di ricerca nel visualizzare i risultati, la

prima volta, non inseriscono il salto pagina. Gli url copiati dovranno essere modificati: sarà necessario inserire tra parentesi quadre la frase e il salto pagina. Per esempio, google avrà i seguenti valori:

- urlRicerca1:

  http://www.google.it/search?q=**[**zucchero+zucchero**]**&hl=it&lr=&ie=UTF-8&oe=UTF-8&start=**[**0**]**&sa=N
- urlRicerca2:

  http://www.google.it/search?q=**[**zucchero+zucchero**]**&hl=it&lr=&ie=UTF-8&oe=UTF-8&start=**[**10**]**&sa=N
- urlRicerca3:

  http://www.google.it/search?q=**[**zucchero+zucchero**]**&hl=it&lr=&ie=UTF-8&oe=UTF-8&start=**[**20**]**&sa=N

In questo modo sarà facilissimo inserire un nuovo motore di ricerca e il sistema provvederà automaticamente a ricavare il salto pagina e la congiunzione.

Query

Query			
idQuery	<pi>	LI	<M>
nomeDominio		VA30	
stringaQuery		VA50	<M>
nrLink		BT3	<M>
pesoTitolo		DC3,3	<M>
pesoDescrizione		DC3,3	<M>
pesoCorpo		DC3,3	<M>
pesoKeyWords		DC3,3	<M>
threshold		DC3,3	<M>
pesoSyG		DC4,2	<M>
pesoSSG		DC4,2	<M>
pesoSeG		DC4,2	<M>
timeOut		I	<M>

Figura 3.13: Query, dettaglio degli attributi

Tra i vari attributi, la maggior parte dei quali derivati dal modello formale, si evincono:

- idQuery: è l'identificativo della query;
- stringaQuery: rappresenta l'insieme di vocaboli da ricercare;
- nomeDominio: indica il dominio di appartenenza della riceca;

- nrLink: rappresenta il numero massimo dei link che deve restituire ogni motore di ricerca;
- timeOut: rappresenta il tempo massimo (in msec) di attesa di una pagina Web.

Link

Il link rappresenta il vero risultato della ricerca. In tale entità vengono memorizzati tutti i voti calcolati. A tale entità si farà riferimento per avere il set di risultati.

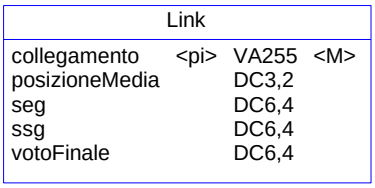

Figura 3.14: Link, dettaglio degli attributi

I voti verranno inseriti dai server in fasi diverse, non necessariamente consecutive. Solo quando sarà stata determinata la posizione media e i voti semantici sarà possibile determinare il voto finale.

L'assegnazione dei voti avviene utilizzando la SKB. Solo in questa entità del repository si riflettono le elaborazioni sintattiche e semantiche sulla pagina web: tali informazioni rappresentano l'interfaccia tra la pagina web e la metrica utilizzata.

Documenti e DocumentiPreProcessati

In tali entità si evincono due attributi importantissimi:

- statoPreProcessor: è un valore booleano e indica, se vero, che il documento è stato preprocessato e vi è una sua occorrenza in DocumentiPreProcessati.
- statoMiner: è un valore booleano e indica, se vero, che il documento è stato oggetto di mining.

Tali attributi permettono ai server di comprendere quali documenti bisogna ancora elaborare.

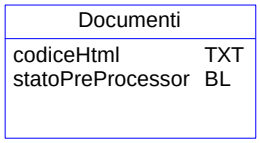

Figura 3.15: Documenti, dettaglio degli attributi

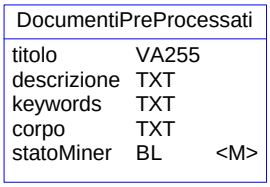

Figura 3.16: DocumentiPreProcessati, dettaglio degli attributi

Stato

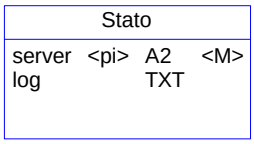

Figura 3.15: Documenti, dettaglio degli attributi

L'attributo server indica il tipo di server: SS per Server Spider, SP per Server Pre-Processor ed SM per Server Miner.

Per ogni query vengono inserite tre occorrenze, ognuna relativa ad un server. Se l'occorrenza è presente significa che il server ha terminato l'elaborazione della query.

Nell'attributo log è possibile inserire informazioni relative ai tempi di elaborazione di una data query, per un dato server. In questo modo è possibile analizzare le prestazioni del sistema e individuare eventuali "colli di bottiglia".

SEMANTIC KNOWLEDGE BASE

La struttura della SKB consente di rappresentare tutte le informazioni relative all'ontologia. Se si pensa al contenuto informativo di una ontologia viene naturale immaginare una SKB abbastanza complessa per poter rappresentare tutte le informazioni linguistiche a svantaggio, però, delle operazioni di mining. Per cui, trascurando alcuni concetti, come le relazioni tra domini e la centralità (per maggiori dettagli si rimanda al capitolo 2) si è potuto creare una SKB abbastanza snella. Inoltre in tale struttura non si è rappresentata la rete semantica ma direttamente la distanza probabilistica tra concetti.

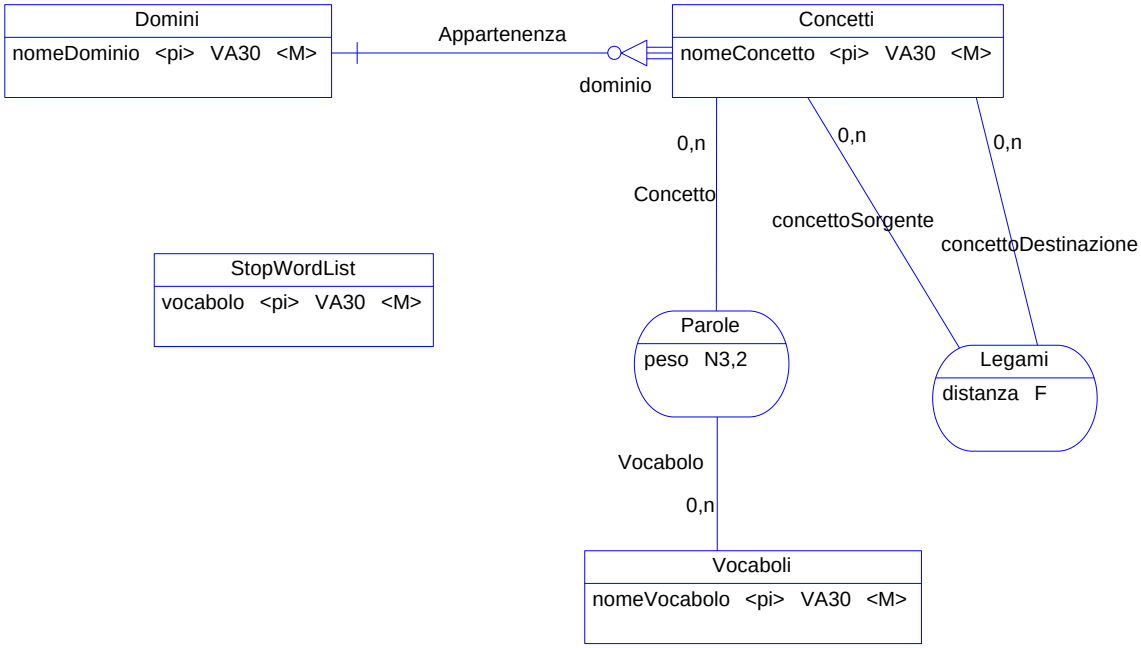

Foto 3.16: schema E-R della SKB

Nel progetto E-R indicato in figura 3.16 si noti la possibilità di gestire più domini e di legare anche concetti con lo stesso nome a domini differenti; questo grazie all'utilizzo di un identificatore esterno per l'entità concetti.

In tale progetto si è anche indicata l'entità StopWordList che contiene tutti i vocaboli non attinenti all'analisi semantica.

E' importante precisare che il sistema e la base dati sono molto flessibili. Per cambiare la metrica del sistema ed utilizzare un'altra base dati è solo necessario modificare la SKB.

Capitolo 4

Tecnologie utilizzate

4.1 Strumenti di sviluppo
4.2 Linguaggio di programmazione
4.3 Oggetti Remoti
4.4 Collegamento ai DataBase: JDBC
4.5 DBMS
4.6 Interazione con il sistema

Nel capitolo precedente abbiamo definito l'architettura del motore di ricerca semantico e abbiamo individuato i componenti essenziali del sistema tra cui, ricordiamo, i tre server che effettuano l'elaborazione. E' stato anche definito il progetto E-R della base dati.

Attualmente, con gli strumenti di case è importante definire alcuni aspetti implementativi, come il linguaggio di programmazione, in modo da avere una corrispondenza con quanto deciso in fase di definizione del progetto e il risultato finale (questo discorso vale in maniera molto forte con il diagramma delle classi).

Coglieremo l'occasione, quindi, per presentare tutte le tecnologie utilizzate: il tipo di DBMS usato e gli strumenti che ci consentono di far interagire il sistema con l'utente.

4.1 STRUMENTI DI SVILUPPO

Per sviluppare l'intero sistema si è usato uno strumento di case che ci ha consentito di realizzare tutti i diagrammi UML e di progettare sia a livello concettuale che a livello fisico la base dati. Tale strumento è Power Designer ed è stato scelto perché pone l'attenzione molto sulla fase progettuale cercando di nascondere quanto più possibile i dettagli implementativi.

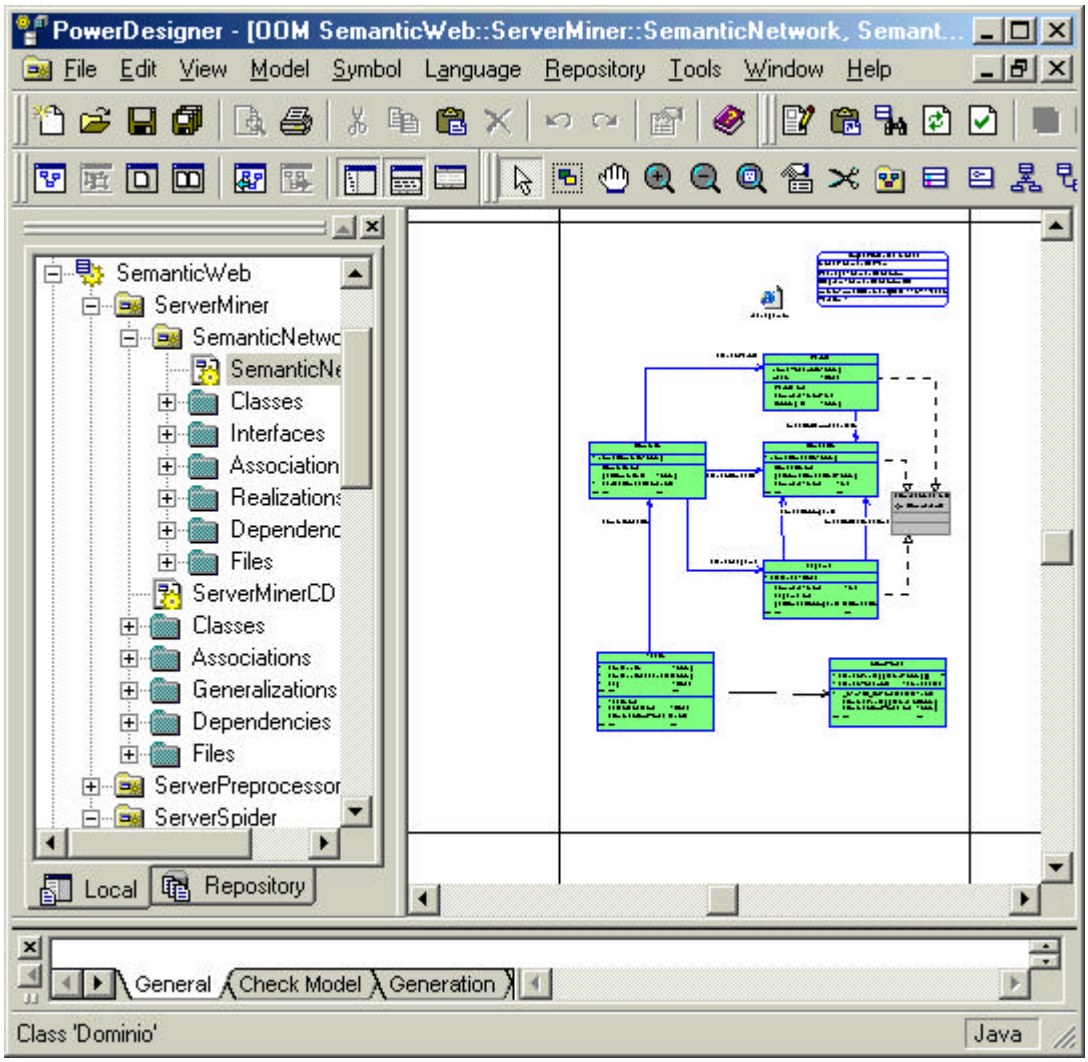

Figura 4.1: utilizzo di Power Designer per progettare il sistema.

Per la realizzazione dell'intero sito web di supporto al sistema è stato utilizzato Dreamweaver 4, uno dei migliori prodotti per lo sviluppo di pagine web.

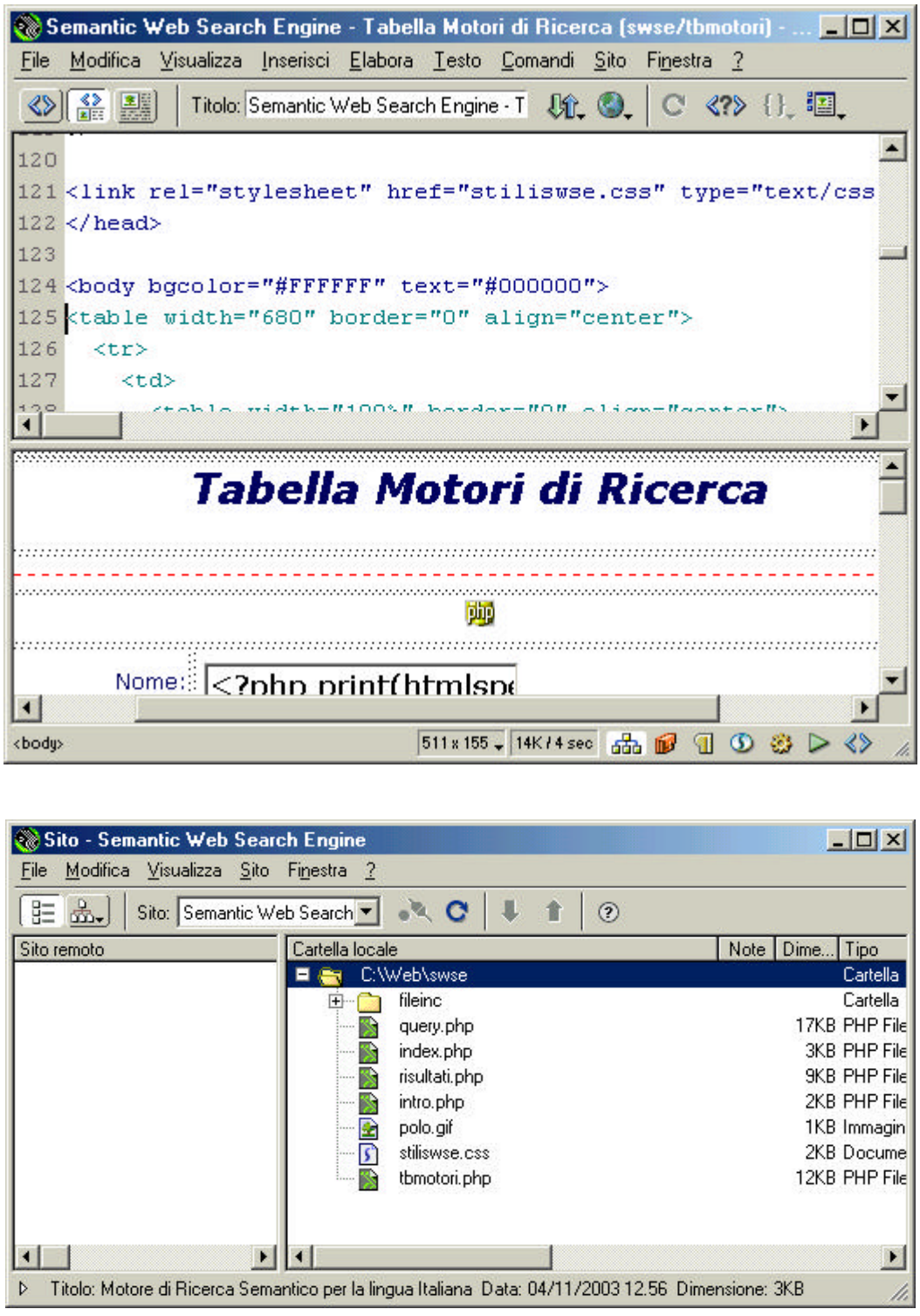

Figura 4.2: utilizzo di DreamWeaver per progettare il sito dinamico.

4.2 LINGUAGGIO DI PROGRAMMAZIONE

Poiché come specifica è richiesto che il sistema sia quanto più possibile indipendente dal sistema operativo e dalla piattaforma hardware come linguaggio di programmazione è stato scelto Java (JDK 1.4).

Inoltre, Java è stato scelto perché è:

- ad oggetti: spinge realmente il programmatore ad utilizzare la tecnologia ad oggetti.
- multithread: è possibile sfruttare tutti i vantaggi della programmazione multithread per aumentare l'efficienza del motore di ricerca semantico.
- distribuito: Java possiede un'estesa libreria di routine per gestire i protocolli TCP/IP. E' possibile accedere in maniera agevole a documenti web e ad oggetti distribuiti (RMI).
- efficiente per il server (servlet): è possibile creare un'applicazione lato server (servlet) in grado di colloquiare con l'utente e di interagire, in maniera distribuita, con altre risorse.
- robusto: Java ha un modello di puntatori che esclude la possibilità di sovrascrivere la memoria e rovinare i dati.

Possiamo quindi sfruttare tutte le caratteristiche di Java a nostro vantaggio e realizzare un sistema che sia:

- indipendente dalla piattaforma
- robusto
- distribuito
- fortemente multithread

E' fortemente multithread perché le singole elaborazioni dei relativi server vengono svolte in maniera concorrente al fine di aumentare l'efficienza totale.

Inoltre, l'interazione con l'utente, per quanto riguarda l'avvio della ricerca, avviene mediante l'utilizzo di una servlet che si comporta da client per il nostro sistema.

La comunicazione tra la servlet e i server avverrà mediante l'utilizzo di Remote Method Invocation (RMI), che permette di gestire ed utilizzare oggetti distribuiti realizzati in Java. Per quanto riguarda RMI, si utilizzerà il meccanismo di caricamento dinamico degli oggetti per far si che eventuali modifiche ai server e sviluppi successivi siano indipendenti dal client (servlet).

4.3 OGGETTI REMOTI

Ci sono periodi in cui la comunità di programmatori pensa di poter risolvere tutti i suoi problemi adottando la filosofia della programmazione a oggetti. L'idea è quella di riuscire a definire una famiglia felice di oggetti che collaborano tra loro e che possono essere posizionati ovunque serva. Logicamente questi oggetti devono poter comunicare attraverso la rete mediante protocolli standard.

Per esempio, ci deve essere un oggetto sul client che l'utente può utilizzare per compilare una richiesta di dati. L'oggetto client invia un messaggio all'oggetto che si trova sul server il quale contiene i dettagli della richiesta. L'oggetto server raccoglie le informazioni richieste, magari accedendo ad un database oppure comunicando con altri oggetti. Quando finalmente è in grado di rispondere alla richiesta, l'oggetto server invia i dati al client.

IL RUOLO DEL CLIENT E DEL SERVER

Per poter chiarire cosa si intende per oggetti remoti, è necessario esaminare l'idea della raccolta locale di informazioni su un computer client e l'invio di dati, attraverso la rete, a un server. Si supponga che, un'utente, su un computer locale, compili un modulo di richieste di informazioni: i dati del modulo vengono inviati al server che elabora la

richiesta e, successivamente, trasmette le informazioni richieste al client, che a sua volta le visualizza, come mostra la figura 4.3.

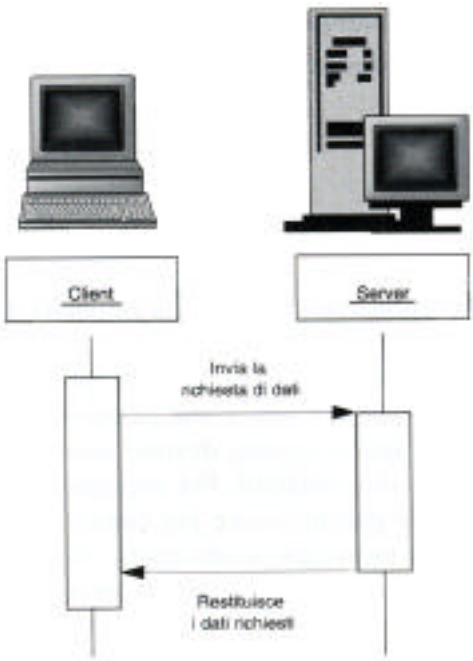

Figura 4.3: la trasmissione degli oggetti tra il client e il server.

Nel tradizionale modello client/server, la richiesta viene tradotta in un formato intermedio (per esempio in coppie nome/valore o in dati XML). Il server analizza il formato della richiesta, elabora una risposta, formatta le informazioni richieste e le trasmette al client. Il client quindi analizza la risposta e la mostra all' utente.

Se però i dati sono strutturati è necessario disporre di un sistema di traduzione appropriato in grado di trasformare i dati dal e nel formato in trasmissione.

La richiesta di dati può essere trasmessa al server anche con i moduli html. In questo caso, il client altri non è che il browser e il server deve raccogliere le informazioni richieste e trasformarle in formato html. Anche in tale modello ha senso separare l'elaborazione del modulo dalla raccolta di dati. In genere la prima operazione avviene sul server web mentre la raccolta dei dati avviene su un altro server chiamato application server. Anche così ci sono due oggetti di comunicazione, l'oggetto client sul server web e l'oggetto server sull'application server. Nell'esporre la tecnologia ci riferiremo al caso tradizionale perché rende più intuitivi i ruoli del client e del srver.

Per risolvere il problema della trasmissione delle informazioni si pensi che gli oggetti che si scambiano le richieste rappresentano le fondamenta dei linguaggi di pro-

grammazione a oggetti (OOP); si possono inserire oggetti su macchine differenti facendo in modo che inviino messaggi direttamente gli uni agli altri. Si supponga che l'oggetto client sia scritto in Java; ciò significa che può essere eseguito, almeno in teoria, ovunque. Per quanto riguarda gli oggetti server ci sono due possibilità:

- l'oggetto non è scritto in Java
- l'oggetto server è scritto in Java

Nel primo caso è necessario disporre di un sistema che consenta agli oggetti di comunicare tra loro nonostante il diverso linguaggio di origine.

Anche solo pensare in via teorica a una cosa del genere risulta naturalmente difficile. In che modo una sequenza di byte scritta in linguaggio arbitrario può indicare i servizi offerti o a quali messaggi risponde? Naturalmente ottenere questo risultato nella pratica non è affatto semplice, anche se l'idea è elegante.

L'architettura CORBA ideata da OMG (Object Management Group, www.omg.org) definisce un meccanismo comune che consente di scambiare dati e scoprire servizi. L'idea di partenza è che l'operazione di individuazione di queste informazioni e attivazione dei servizi richiesti deve essere delegata ad un ORB (Object Request Broker). L'ORB è una specie di traduttore universale che si occupa della comunicazione tra oggetti. Gli oggetti non comunicano sempre tra loro ma utilizzano sempre come intermediario un ORB. Gli ORB sono sparsi sulla rete ed è importante che possano comunicare gli uni con gli altri. La maggior parte degli ORB segue le specifiche impostate da OMG per la comunicazione tra ORB. Queste specifiche sono chiamate IIOP (Internet Inter-ORB Protocol).

Per gestire la comunicazione tra oggetti, Microsoft utilizza un diverso protocollo di basso livello chiamato COM. I servizi simili a ORB sono integrati nel sistema operativo Windows. Non più tardi del 1999 COM è diventato il principale concorrente di CORBA, anche se al momento Microsoft ha deciso di passare al modello .NET, progettato per far in modo che i componenti software comunicassero tra loro basandosi su XML.

CORBA è un linguaggio completamente neutrale. I programmi client e server possono essere scritti in C++, Java o qualsiasi linguaggio compatibile con CORBA. Per specificare le firme dei messaggi e i tipi di dati che gli oggetti possono inviare e comprendere si utilizza un IDL (Interface Definition Language). Le specifiche IDL assomigliano molto alle interfacce Java; infatti è come definire le interfacce che gli oggetti in comunicazione devono supportare. Una caratteristica di questo modello è la possibilità

di fornire una specifica IDL per un oggetto ereditario esistente e successivamente accedere ai sui servizi attraverso ORB anche se è stato scritto molto tempo prima del suo avvento.

Pochi ritengono che CORBA sia il modello di oggetto del futuro e che Java sia lo strumento ideale per implementare client e server CORBA. Per la verità corba è stato più volte criticato, qualche volta meritatamente, per la sua lentezza, per la complessità delle sue implementazioni e i suoi problemi di interoperabilità.

Se entrambi gli oggetti di comunicazione sono scritti in Java, non è necessario affidarsi alla complessità di CORBA. Sun ha sviluppato un meccanismo più semplice chiamato RMI (Remote Method Invocation) più adatto alla comunicazione tra applicazioni Java.

I sostenitori di CORBA inizialmente non apprezzavano RMI perché ignorava completamente lo standard CORBA. Tuttavia, a partire da SDK 1.3, CORBA e RMI sono diventati più interoperativi. In particolare è possibile utilizzare RMI con il protocollo IIOP al posto del proprietario Java Remote Method Protocol, tuttavia, mentre CORBA sarà più interessante in futuro, RMI è più semplice da comprendere e più facile da utilizzare. Va comunque ricordato che RMI è utile solo nella comunicazione tra oggetti Java.

REMOTE METHOD INVOCATION

Il meccanismo RMI permette di fare qualcosa che sembra semplice. Se si può accedere ad un oggetto che si trova su un'altra macchina è possibile chiamare i metodi dell'oggetto remoto. Naturalmente i parametri del metodo devono essere comunicati in qualche modo all'altra macchina, all'oggetto deve essere chiesto di eseguire il metodo e il valore restituito deve essere inviato al client. RMI gestisce tutti questi dettagli.

Per esempio, il client che chiede informazioni su un certo prodotto può interrogare un oggetto Warehouse che si trova su server. Chiama il metodo remoto Find che usa un parametro: l'oggetto Customer. Il metodo Find restituisce al client un oggetto: quello delle informazioni relative al prodotto (figura 4.4).

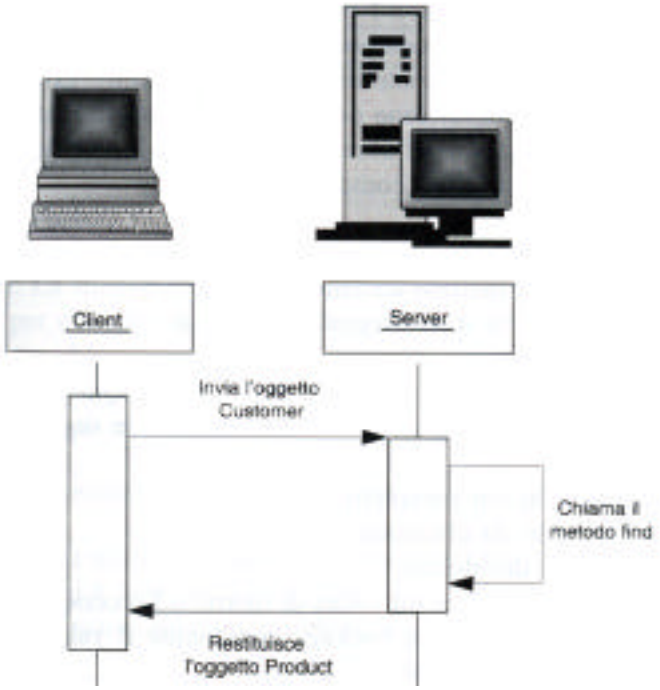

Figura 4.4: la chiamata ad un metodo remoto su un oggetto server.

Nella terminologia adottata da RMI, l'oggetto il cui metodo crea la chiamata remota viene definito oggetto client. L'oggetto remoto è chiamato oggetto server. È importante ricordare che la terminologia client/server si applica solo a un singola chiamata di metodo. Il computer che esegue il codice Java chiama il metodo remoto che rappresenta il client di quella chiamata e il computer che ospita l'oggetto che elabora la chiamata rappresenta Il server di quella chiamata. È possibile che i ruoli cambiano durante la comunicazione. Il server di un precedente chiamata può diventare il client quando chiama un metodo remoto su un oggetto che si trova su un alto computer.

STUB E MARSHAL DEI PARAMETRI

Quando desidera chiamare un metodo remoto che si trova su un oggetto remoto, in realtà il codice client chiama un metodo ordinario di Java incapsulato in un oggetto surrogato detto anche stub. Lo stub si trova sulla macchina client, non sul server. Lo stub raccoglie i parametri usati nel metodo remoto in un blocco byte. Questa operazione utilizza per ciascun parametro una codifica indipendente dal dispositivo per ogni parametro. Per esempio, i numeri vengono sempre inviati con l'ordinamenti di byte big-endian.

Gli oggetti sono codificati con il meccanismo di serializzazione. Il processo di codifica dei parametri viene chiamato marshal dei parametri. Lo scopo del marshal dei parametri è la conversione dei parametri in un formato adatto al trasporto da una virtual macchine all'altra. Riepilogando: il metodo stub sul client crea un blocco di informazioni contenente:

- ✓ un identificatore dell' oggetto remoto da utilizzare;
- ✓ una descrizione del metodo da chiamare;
- ✓ i parametri sui quali è stato eseguito il marshal.

Lo stub invia quindi queste informazioni al server. Qui un oggetto ricevitore esegue per ciascuna chiamata al metodo remoto le seguenti azioni:

- ✓ esegue l'unmarshal sui parametri;
- ✓ individua l'oggetto da chiamare;
- ✓ chiama il metodo desiderato;
- ✓ cattura ed esegue il marshal sul valore di ritorno o l'eccezione della chiamata;
- ✓ invia allo stub del client un package contenente il valore di ritorno sul quale è stato eseguito il marshal.

Lo stub del client esegue l'unmarshal sul valore di ritorno o sul eccezione proveniente dal server. Questo valore diventa il valore restituito dalla chiamata allo stub. Oppure, se il metodo remoto lancia un'eccezione lo stub la rilancia nello spazio di elaborazione del chiamante. La figura 4.5 mostra il flusso di informazioni di una chiamata al metodo remoto.

Questo processo è ovviamente complesso, ma l'aspetto positivo è che è completamente automatico e trasparente per il programmatore. Inoltre, i progettisti degli oggetti remoti di Java hanno cercato di farli assomigliare agli oggetti locali. La sintassi per una chiamata al metodo remoto è uguale a quella utilizzata per un chiamata locale.

Quindi, per utilizzare oggetti remoti è necessario creare e rendere disponibili delle opportune interfacce. Naturalmente le interfacce sono entità astratte che indicano espressamente quali metodi possono essere chiamati insieme alle loro firme. Variabili di tipo interfaccia devono sempre essere associate a un oggetto reale di qualche tipo.

Quando si chiamano i metodi remoti, la variabile oggetto fa riferimento a un oggetto stub. Il programma client non conosce realmente il tipo di questi oggetti. Le classi stub e

gli oggetti associati vengono creati automaticamente. Anche se gli sviluppatori sono riusciti a nascondere molto bene ai programmatoti numerosi dettagli della chiamata al metodo remoto, la procedura è alquanto complessa.

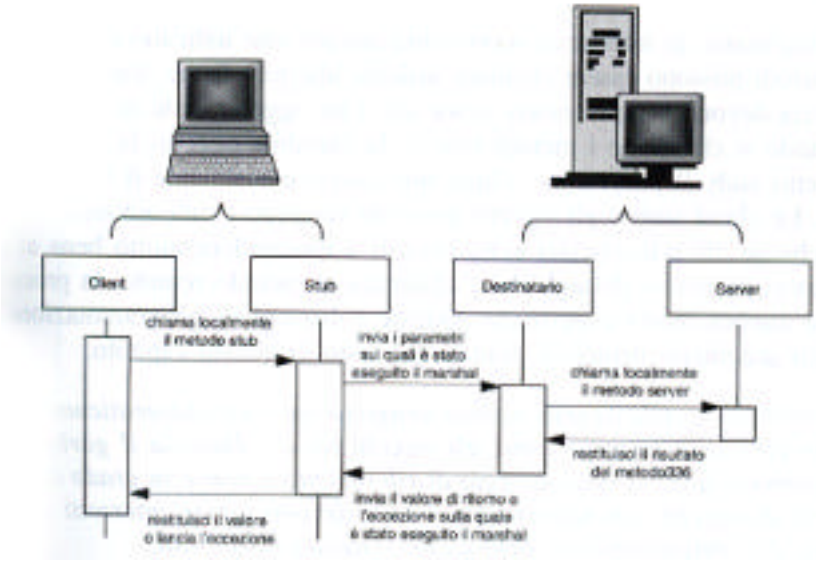

Figura 4.5: il marshal dei parametri.

CARICAMENTO DI CLASSE DINAMICO

Quando si passa un oggetto remoto ad un altro programma, sia come parametro si come valore di ritorno di un metodo remoto, il programma deve potersi occupare dell'oggetto stub associato. Ciò significa che deve avere il codice per la classe stub. I metodi stub non svolgono un lavoro molto interessante: non fanno altro che eseguire il marshal e l'unmarshal sui parametri e quindi contattare il server per le chiamate al metodo. Tutto ciò viene eseguito in modo trasparente al programmatore.

Inoltre, anche le classi per i parametri, per i valori di ritorno e per gli oggetti eccezione possono dover essere caricate. Il caricamento può essere più complesso del previsto. Per esempio, si può dichiarare un metodo remoto con un certo tipo di dati di ritorno al client, ma il metodo può in realtà restituire un oggetto di una sottoclasse sconosciuta al client. Il caricatore di classe caricherà quindi la classe derivata.

Le classi stub devono essere a disposizione del programma client in esecuzione. Lo si può fare inserendo le classi nel file system locale. Tuttavia se il programma server è esteso e vengono aggiunte nuove classi per i tipi di ritorno e per le eccezioni, può risultare impossibile mantenere il client aggiornato.

Per questo motivo i client RMI possono caricare automaticamente le classi stub da un altro posto. Il processo è simile all'operazione di caricamento di classi delle applet eseguite in un browser.

Ogni volta che un programma carica un nuovo codice da un'altra posizione sulla rete si presenta un problema di sicurezza. Per questo motivo è necessario adottare un gestore di sicurezza nelle applicazioni client RMI. Il meccanismo di sicurezza protegge il programma dal virus che potrebbero essere presenti nel codice stub. Per applicazioni specializzate, i programmatori possono adottare caricatori di classi e gestori di sicurezza personalizzati, anche se quelli forniti dal gestore RMI sono sufficienti per un normale utilizzo.

4.4 COLLEGAMENTO AI DATABASE: JDBC

Nell' estate del 1996 Sun ha rilasciato la prima versione del pacchetto Java Database Connectivity (JDBC). Questo pacchetto permette ai programmatori di collegarsi a un database, interrogarlo o aggiornarlo utilizzando Structured Query Language.

Probabilmente si tratta dello sviluppo più importante della piattaforma Java, anche perché i database rappresentano attualmente l'applicazione informatica più comune. Java e JDBC offrono il seguente vantaggio: i programmi sviluppati in linguaggio Java e JDBC sono indipendenti dalla piattaforma e dal produttore.

Lo stesso programma di database scritto in Java può funzionare su una macchina NT, su un server Solaris o un dispositivo potenziato dalla piattaforma Java. È possibile portare i dati da un database a un' altro, per esempio da Microsoft SQL Server a Oracle, o addirittura a un piccolo database incorporato in un apparecchio, e lo stesso programma può ancora leggere gli stessi dati. Tutto questo è in netto contrasto con la tradizionale programmazione di database. Capita ancora troppo spesso che si scrivano applicazioni database in linguaggi di database proprietari utilizzando un sistema di gestione dei dati fornito da un unico produttore. Ciò che si ottiene alla fine è un'applicazione database

che può essere utilizzata solo su una o al massimo due piattaforme. È perciò molto probabile che il linguaggio Java e JDBC, a causa della loro universalità, alla fine riescano a sostituire i linguaggi di database proprietari e le interfacce a livello chiamata utilizzate da Orcale, Informix e Microsoft per l' accesso ai database. A prova di ciò si pensi che alcun produttori di database ora creano una JVM nel database stesso in modo da permettere al programmatore di scrivere in Java le procedure memorizzate. Questa tecnologia fa parte di una specifica SQLJ; per ulteriori informazioni consultare il sito http: / /www.sqlj.org.

Insieme alla versione Java 2, presentata nel 1998, è stata proposta anche una seconda versione JDBC. Il supporto JDBC2 non è ancora universale ma sono disponibili molti driver JDBC2. JDBC2 introduce molte funzioni importanti, come i cursori scorrevoli e il supporto per tipi SQL avanzati, mentre JDBC 3 introduce miglioramenti di natura tecnica.

LA STRUTTURA DI JDBC

Fin dall'inizio gli sviluppatori della tecnologia Java di Sun conoscevano il potenziale dimostrato da Java nel lavoro con i database. A partire dal 1995 cominciarono a estendere la libreria standard Java pensando a SQL. Inizialmente pensarono di estendere Java in modo da permettergli di comunicare con qualsiasi database utilizzando solo il linguaggio puro. Ben presto, però, si accorsero che sarebbe stato impossibile raggiungere questo obbiettivo, in quanto esistevano troppi database associati a troppi protocolli diversi. Inoltre, anche se accettavano il fatto che Sun lavorasse alla scelta del protocollo di rete standard dedicato all'accesso di database, i produttori di database erano favorevoli solo alla scelta del proprio protocollo di rete.

I produttori di database e degli strumenti erano d'accordo solo su un fatto: sarebbe stato meglio che Sun avesse fornito API in Java puro per l'accesso SQL insieme ad uno gestore di driver capace di consentire a driver di terze parti di collegarsi a database specifici. I produttori avrebbero fornito i propri driver da unire al gestore di driver. Sarebbe stato necessario adottare un semplice meccanismo in grado di registrare i driver di terze parti nel gestore di driver; i driver non avrebbero dovuto far altro che seguire i requisiti definiti nell' API del gestore di driver.

Dopo un lungo periodo di discussioni pubbliche, l'API per l'accesso ai database divenne l'API JDBC e le regole di scrittura del driver vennero incapsulate nell'interfaccia del fornitore di servizi JDBC.

Questo protocollo segue il modello di successo di ODBC di Microsoft, che fornisce un'interfaccia nel linguaggio di programmazione C per l' accesso ai database. Sia JDBC sia ODBC si basano sulla medesima idea: i programmi scritti secondo le specifiche dell'API JDBC sono in grado di comunicare con il gestore del driver JDBC che, a sua volta, utilizza i driver registrati per comunicare con il database.

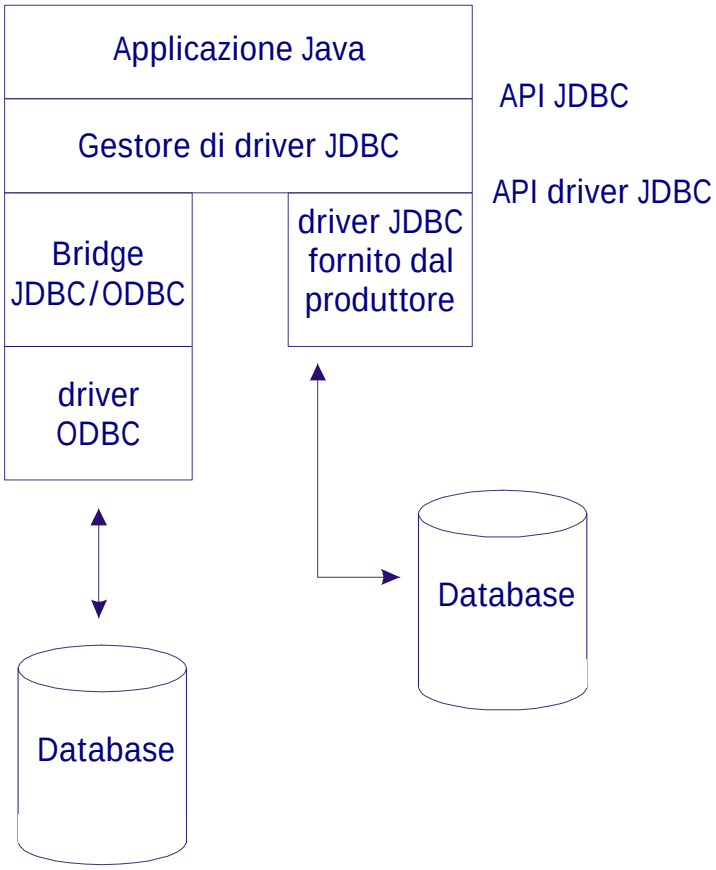

Figura 4.6: comunicazione tra JDBC e un database.

Più precisamente, JDBC è costituito da due livelli. Il livello superiore è l'API JDBC. Quest'API comunica con l'API del driver di gestione JDBC al quale invia le diverse dichiarazioni SQL.

Il gestore dovrebbe comunicare (in modo trasparente al programmatore) con i vari driver di terze parti che si collegano al database e restituire le informazioni ottenute dalla query o eseguire l'azione specifica della query.

La specifica JDBC consente di passare qualsiasi stringa al driver sottostante. Il driver può passare questa stringa al database. Questa caratteristica consente di utilizzare versioni specializzate di SQL che possono essere supportate dal driver e dal suo database associato. Tutto questo significa che il livello Java/JDBC è tutto ciò che i programmatori, nella maggior parte dei casi, devono considerare.

I driver JDBC sono raggruppati in quattro tipi:

- I driver di tipo 1:

  traducono JDBC in ODBC e si affidano al driver ODBC per comunicare con il database. Sun fornisce in JDK un driver di questo tipo chiamato JDBC/ODBC bridge. Tuttavia il bridge richiede l' utilizzo e la configurazione corretta di un driver ODBC. Quando è stato rilasciato JDBC, il bridge era utile per i test, ma non è mai stato pensato per uso produttivo. Attualmente ci sono tantissimi driver disponibili e si consiglia di utilizzare il bridge JDBC/ODBC.

- Il driver di tipo 2:

  sono scritti parte in Java e parte in codice nativo e comunicano con l'API client di un database. Quando si implementa tale driver, oltre a una libreria Java è necessario installare un codice specifico per la piattaforma.

- Il driver di tipo 3:

  sono librerie client Java pure e utilizzano un protocollo indipendente dal database per trasmettere le richieste a un componente server che la traduce nel protocollo specifico del database. La libreria è indipendente dal database reale, e questo ne semplifica l' uso.

- I driver di tipo 4:

  sono librerie Java pure e traducono le richieste JDBC direttamente nel protocollo specifico del database.

La maggior parte dei produttori del database fornisce insieme al database anche driver di tipo 3 e 4. Inoltre diverse società sono specializzate nella produzione di driver che oltre ad essere maggiormente conformi agli standard supportano anche un maggior nu-

mero di piattaforme o, in alcuni casi, sono semplicemente più affidabili dei driver forniti dai produttori di database.

Per concludere, l'**obbiettivo finale di JDBC è**:

- ✓ dare ai programmatori la possibilità di scrivere applicazioni in Java capaci di accedere ad ogni database utilizzando dichiarazioni SQL standard (o anche estensioni specializzate di SQL), seguendo comunque le convenzioni Java (tutti i driver JDBC devono supportare almeno la versione SQL 92);

- ✓ permettere ai produttori di database e di strumenti per database di fornire i driver di basso livello. In tal modo essi possono ottimizzare i driver per i loro prodotti specifici.

Sun non ha adottato il modello ODBC per i seguenti motivi (le affermazioni sono state registrate durante la conferenza JavaONE tenuta nel mese di maggio del 1996);

- ✓ ODBC è difficile da imparare;

- ✓ ODBC ha pochi comandi ma moltissime opzioni complesse. Lo stile preferito nel linguaggio Java si basa sui molti metodi semplici e intuitivi;

- ✓ ODBC fa uso dei puntatori void e altre funzionalità del linguaggio C che non sono naturali in Java;

- ✓ una soluzione basata su ODBC è meno sicura e più difficile da realizzare di una soluzione Java pura.

UTILIZZO TIPO DI JDBC

È possibili utilizzare JDBC sia nelle applicazioni sia nelle applet. In un'applet vengono applicate tutte le normali restrizioni di sicurezza. Per impostazione predefinita il gestore della sicurezza presuppone che tutte le applet scritte in Java non siano affidabili. In particolare, le applet che utilizzano JDBC possono solo aprire una connessione database verso il server dal quale sono state prelevate. Ciò significa che il server web e il server database devono essere la stessa macchina, configurazione abbastanza singola-

re. Naturalmente il server web può utilizzare un servizio proxy per istradare il traffico di informazioni legate al database verso un'altra macchina. Nel caso di applet firmate, questa restrizione può essere annullata.

Le applicazioni, d'altra parte, sono completamente libere di accadere a server di database remoti. Se si implementa un tradizionale programma client/server (figura 4.4), per accedere al database probabilmente è più logico adottare un' applicazione e non un'applet .

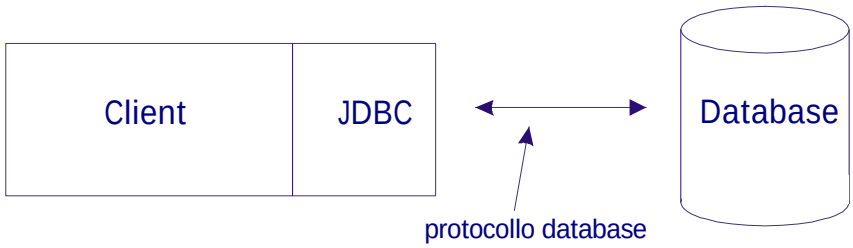

Foto 4.7: un'applicazione client-server

Tuttavia il mondo si sta spostando dalla filosofia client/server verso un modello a tre tier se non addirittura verso i più avanzati modelli a n tier. Nel modello a tre tier il client non genera le chiamate indirizzate al database ma chiama un livello intermedio che si trova sul server che si occupa di creare le query. Il modello a tre tier ha un paio di pregi: separa la presentazione visiva (sul client) dalla logica dell'operazione (nel livello intermedio) e dai dati grezzi (del database). Quindi diventa possibile accedere ai medesimi dati e alle medesime regole da più client, come per esempio un'applicazione Java, un applet o um modello web.

La comunicazione tra client e tier intermedio può avvenire attraverso HTTP (quando si utilizza un broswer web come client), RMI o un'altro meccanismo. JDBC è utilizzato per gestire la comunicazione tra il tier intermedio e la parte finale del database. La figura 4.5 mostra l' architettura di base.

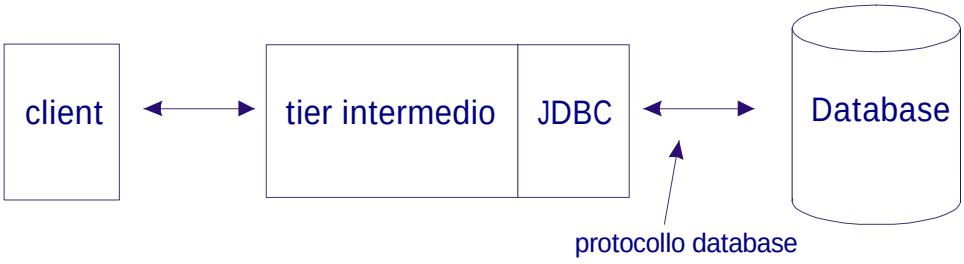

Figura 4.8: un'applicazione a tre tier.

Naturalmente questo modello può avere diverse varianti. In particolare, Java 2 Enterprise Edition definisce un struttura per application server che gestiscono moduli di codice chiamati Enterprise JavaBeans, e fornisce servizi utili come per esempio il bilanciamento del carico, la memorizzazione in cache delle richieste, la gestione di sicurezza e l'accesso al database. In questa architettura JDBC gioca ancora un ruolo importante nella creazione di query complesse (per ulteriori informazioni su interprise edition consultare http: / / Java.sun.com/j2ee).

4.5 DBMS

Il DBMS utilizzato è di tipo relazionale e la comunicazione con la base dati del sistema avviene utilizzando il linguaggio SQL-2, attenendosi completamente allo standard. Come RDBMS è stato scelto MySQL perché:

- è open source
- compatibile con lo standard SQL-2
- la select è molto efficiente

Inoltre, MySQL è supportato, in modo nativo, da diversi linguaggi di scripting lato server. In questo modo gli script lato server che accedono alla base dati possono essere realizzati in modo molto efficiente.

4.6 INTERAZIONE CON L'UTENTE

L'utente interagisce con il sistema mediante il web. E' quindi necessario fare in modo che questa interazione sia il più efficiente possibile.

Ogni informazione che l'utente o l'amministratore vorrà inviare al sistema dovrà essere controllata sul client, evitando così di sovraccaricare il Web Server ed evitando

all'utente inutili attese e ripetuti tentativi di comunicazione. Come linguaggio di scripting lato client si è utilizzato JavaScript 1.2, che è compatibile con tutte le ultime versioni dei browser presenti sul mercato. Inoltre il codice JavaScript inserito è conforme allo standard ECMA.

Abbiamo evidenziato nel capitolo precedente che molte operazioni vengono eseguite dagli script lato server. Tali operazioni riguardano:

- la memorizzazione della query nella base dati;
- la visualizzazione dei risultati;
- la gestione del sistema da parte dell'amministratore.

Per permettere che tali operazioni fossero eseguite in maniera efficiente si è utilizzato PHP 4 come linguaggio di scripting lato server che da la possibilità di interagire direttamente con MySQL.

Capitolo 5

Sviluppo del Sistema

5.1 Package Util
5.2 Server Spider
5.3 Server Preprocessor
5.4 Semantic Network
5.5 Server Miner

E' stato possibile sviluppare il sistema prevedendo la definizione di una serie di diagrammi di classi, ciascuno per ogni server (package). Poiché abbiamo strutturato il sistema in modo da far sincronizzare i vari server nel database, con il salvataggio dello stato, lo sviluppo dei server è stato realizzato in maniera indipendente. In una seconda fase, poi, è stata sviluppata l'intera interfaccia di comunicazione (presentata nel capitolo successivo).

Nel presentare lo sviluppo del sistema, oltre ad indicare i vari diagrammi delle classi, sono esposti diagrammi di sequenza e diagrammi di attività per mostrarne il funzionamento.

Per ogni classe si indicano le principali funzionalità, in termini di messaggi pubblici che è possibile utilizzare, tralasciando la spiegazione dei dettagli implementativi, per i quali si rimanda alle relative API (allegato).

5.1 PACKAGE UTIL

Ogni applicazione prevedere un insieme di classi che sono di supporto all'esecuzione di particolari elaborazioni. Il package Util ha il compito di accedere a risorse esterne al sistema e che possono essere rese disponibili ai vari server. Tali risorse permettono di calcolare i tempi di elaborazione, di accedere alle pagine web e di elaborarle, e di accedere ad una base dati tramite un particolare RDBMS.

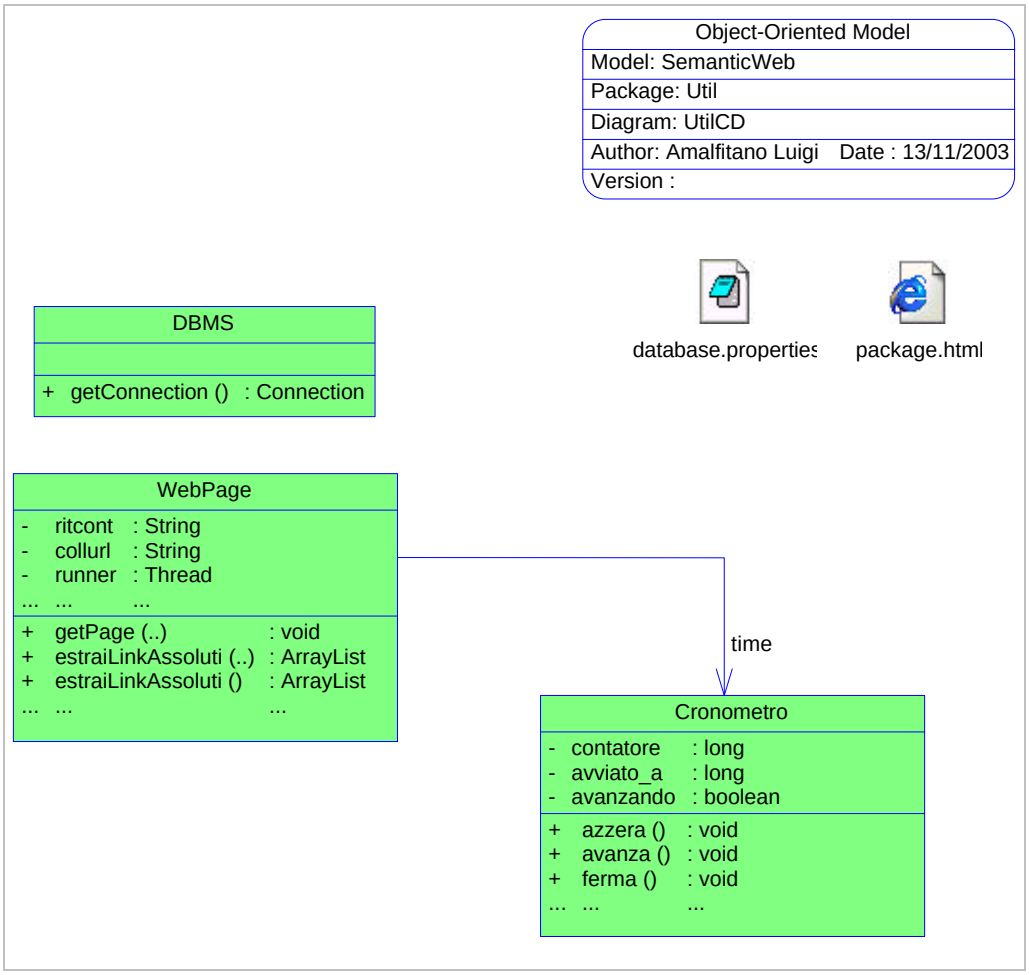

Figura 5.1: ecco come si presenta il package Util in PowerDesigner

Il file package.html ha lo scopo di documentare il package corrente e il file database.properties è utilizzato dalla classe DBMS.

LA CLASSE CRONOMETRO

La classe cronometro ha il compito di gestire tutti i metodi che permettono di misurare i tempi di elaborazione.

Foto 5.2: la classe cronometro del sistema

I metodi di tale classe permettono di azzerare l'oggetto cronometro, farlo avanzare da un determinato punto, farlo avanzare da capo e leggerne il valori. Per maggiori dettagli si rimanda all'API relativa.

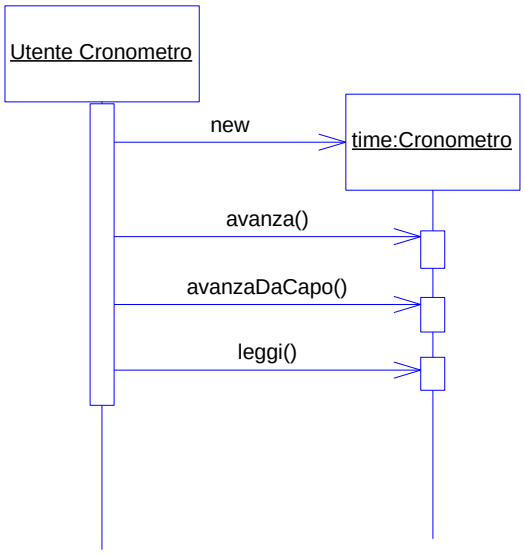

Foto 5.3: possibile utilizzo di un oggetto della classe.

LA CLASSE WEBPAGE

La classe WebPage permette di accedere alle pagine web e di estrarne i link: rappresenta il WEB CATCHER e il PARSER del sistema.

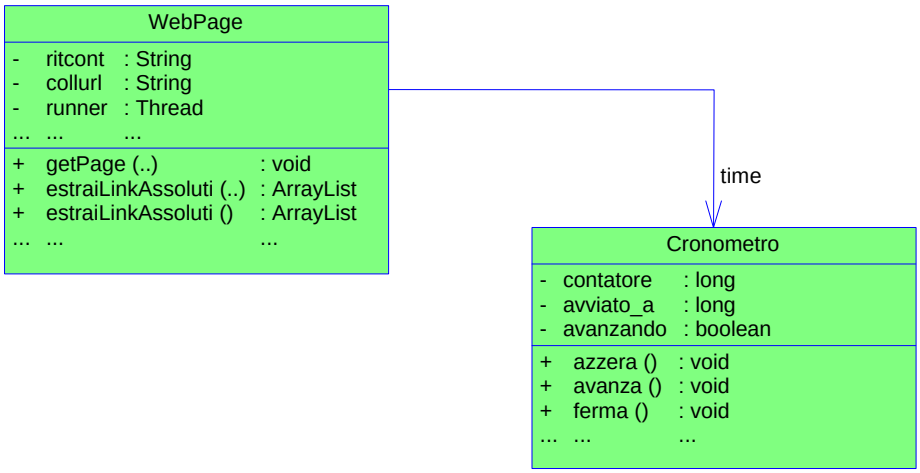

Figura 5.4: la classe WebPage

Il prelievo di una pagina Web associata ad un indirizzo http (Web Catcher) viene effettuato dal metodo getPage(). E' importante precisare che la richiesta di una pagina web provoca l'attesa del Thread corrente fino a quando l'operazione di lettura non termina.

Il flusso di dati provenienti da una pagina web può non essere disponibile, causando il blocco del Thread corrente, anche per un periodo lungo (è il sistema operativo che tiene sotto controllo i flussi I/O).

Per questo motivo è stato necessario implementare sul flusso proveniente dalla pagina web un TimeOut che è possibile passare al metodo getPage().

Per fare ciò si genera un nuovo Thread (quello che realmente si preoccupa della connessione e del reperimento delle informazioni). Tale Thread verrà avviato e poi ci si porrà in attesa che l'operazione vada a buon fine o scada il TimeOut e il Thread corrente verrà così riavviato.

I link presenti all'interno della pagina web possono essere estratti (Parser) solo se ne è stato prelevato il contenuto. E' inoltre possibile applicare un filtro che consente di scartare link a cui è associato un determinato valore.

Il metodo estraiLinkAssoluti() consente di:

- ✓ analizzare tutto il contenuto html;
- ✓ applicare eventualmente un filtro;
- ✓ valutare il tag <a>;
- ✓ analizzarne i valori degli attributi che contengono eventuali collegamenti assoluti, e stabilire quale considerare: viene data priorità agli eventi di tipo javascript;
- ✓ individuare link conformi alle specifiche http;
- ✓ scartare eventuali duplicati, se presenti.

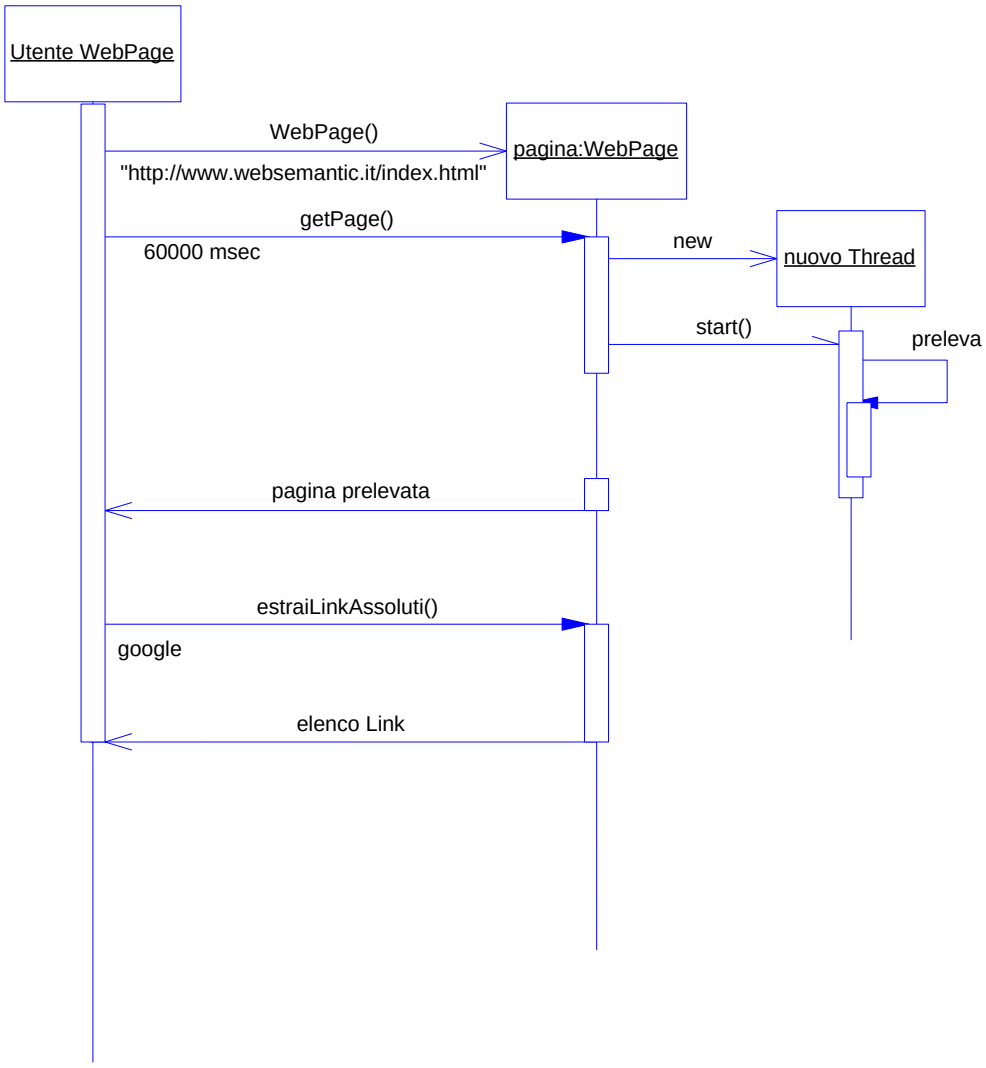

Figura 5.5: possibile utilizzo della classe

Nel diagramma di sequenza si noti che l'utente attende che venga prelevata la pagina da internet. Tale compito è demandato ad un nuovo thread che restituirà il risultato nel

contesto dell'oggetto pagina, che, intanto, è sospeso: viene riattivato alla fine della lettura della pagina o allo scadere del timeout (in msec) impostato.

Inoltre i link sono filtrati con la parola google: tutti i tag che hanno un collegamento http e che contengono questa parola vengono scartati.

LA CLASSE DBMS

La classe DBMS gestisce la connessione al server DataBase: poiché viene utilizzato Java come linguaggio di programmazione si utilizza Java Database Connectivity – JDBC per la connessione al database (per maggiori chiarimenti su JDBC si rimanda al capitolo 4).

I dati di connessione vengono prelevati da un file di proprietà (database.properties) presente nella stessa cartella della classe. In tal modo sarà possibile cambiare sia l'RDBMS che il database del sistema senza necessariamente dover modificare il codice.

Nel file database.properties viene specificato il driver JDBC e i vari parametri di connessione:

- jdbc.drivers=[driver jdbc]

  indica il driver jdbc da utilizzare, dipende dal particolare RDBMS

- jdbc.hostname=[host e porta secondo la sintassi jdbc]

  rappresenta l'indirizzo su cui è connesso il server DB

- jdbc.dbname=[nome del DB]

  rappresenta il nome del database del sistema

- jdbc.username=[utente]

  rappresenta l'utente che ha l'accesso al sistema. L'accesso deve prevedere la possibilità di effettuare select e insert.

- jdbc.password=[password]

  è la password assegnata all'utente.

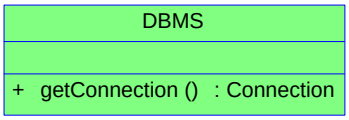

Figura 5.6: classe che gestisce la connessione al databse

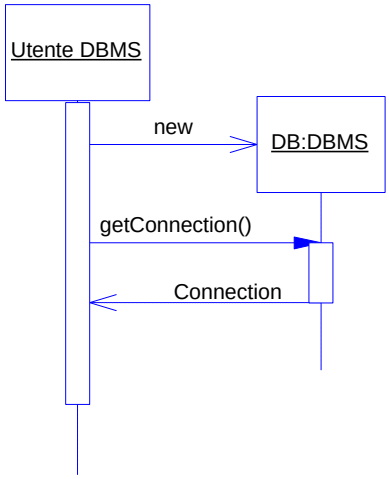

Figura 5.7: classe che gestisce la connessione al databse

Le classe prevede un unico metodo, pubblico, getConnection() che ha il compito di effettuare la connessione. Restituisce un oggetto (di tipo Connection) che rappresenta la connessione effettuata al sistema.

5.2 SERVER SPIDER

Il progetto del server spider è concentrato all'interno del package ServerSpider. In tale package vengono sviluppate tutte le funzionalità relative al Search Engine Wrapper e allo Spider Agent.

Tale package è composto dalle seguenti classi:

- Link:

  Le istanze di questa classe rappresentano i link individuati. Ogni link è associato ad un oggetto Query e a un array contenete i motori che lo hanno individuato.

  Dispone di un WEB CATCHER ed implementa REPOSITORY BUILDER.

- Motori Di Ricerca:

  E' la classe le cui istanze gestiscono l'interrogazione ai motori di ricerca. Ogni oggetto di tale classe si riferisce ad un particolare motore di ricerca che deve essere presente nel DataBase del sistema.

  Implementa il QUERY ADAPTER, il SEARCH ENGINE SUBMITTER e dispone di un PARSER.

- Query:

  E' la classe che fa riferimento alle query di cui bisogna fare lo spidering. Si suppone che le query siano già state archiviate nel DB da altri moduli. L'operazione di definizione dell'oggetto query si completa con il caricamento dei dati dal DB del sistema.

  Ha il compito di gestire le operazioni e di organizzare l'elenco dei link

Nella figura 5.8 è riportato il diagramma delle classi relative allo Spider.

Si evidenzia l'associazione MotoriScelti della classe Query con la classe MotoriDiRicerca, necessaria per poter gestire i motori di ricerca selezionati per la query data. Sarà l'oggetto Query a gestire l'elenco degli oggetti motori di ricerca.

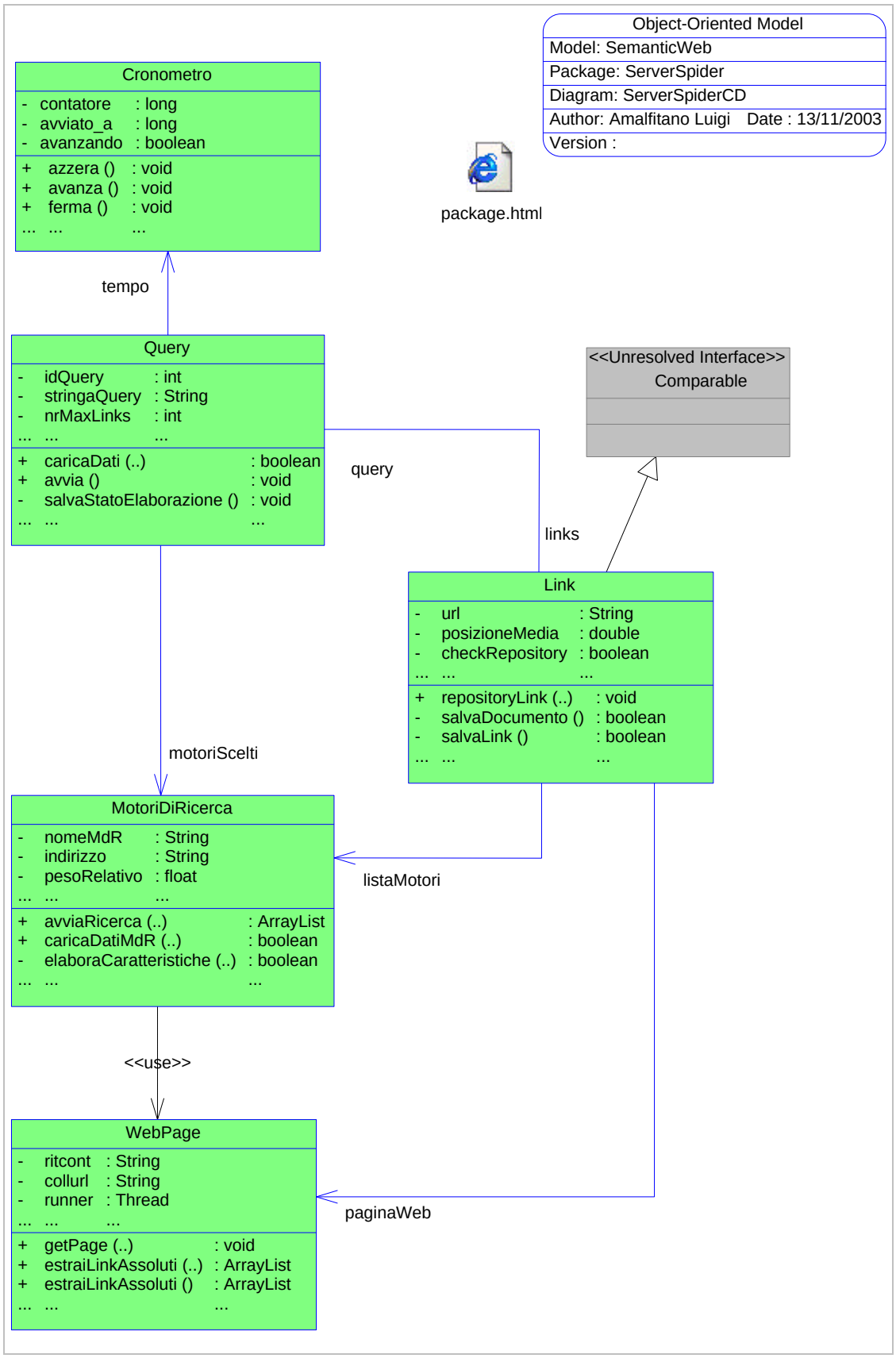

Figura 5.8: ecco come si presenta il package ServerSpider in PowerDesigner

Inoltre vi è l'associazione tra la Query e la classe Link. Tale associazione è bidirezionale e permette di associare ad ogni Link individuato la Query a cui si riferisce. Sarà compito dell'oggetto Query creare e definire gli oggetti Link.

Vi è poi l'associazione listaMotori che permette all'oggetto Link di mantenere informazioni sui motori di ricerca che lo hanno individuato.

Notiamo che la classe Link è associata alla classe WebPage: questo è dovuto al fatto che ogni link si riferisce ad una determinata pagina Web, di cui bisogna prelevarne il contenuto (Web Catcher).

Inoltre, la classe MotoriDiRicerca fa uso della classe WebPage, questo perché i servizi offerti da un determinato motore di ricerca sono disponibili su una o più pagine web, a cui è necessario avere accesso (Parser).

Alla classe Query è associata, poi, la classe Cronometro. Questo ci consente di calcolare i tempi di elaborazione.

Infine si osservi che, il file package.html è necessario per la documentazione.

Nei paragrafi successivi mostreremo le caratteristiche di ciascuna classe e alla fine indicheremo il funzionamento totale del sistema.

LA CLASSE LINK

Gli oggetti della classe link si riferiscono ai collegamenti http individuati dal Parser all'interno delle pagine risultato restituite dai motori di ricerca.

All'atto dell'individuazione di un nuovo link viene creato un oggetto link ed immediatamente avviatone il download della pagina, ovvero il Repository Builder.

Questo è possibile poiché ogni link è associato ad un oggetto WebPage: all'atto della creazione, l'oggetto link crea un oggetto WebPage con lo stesso collegamento http.

Con l'invio del messaggio repositoryLink() si individua la relativa pagina web e se ne preleva il contenuto (operazione eseguita dal Web Catcher, ovvero dall'oggetto WebPage). Se la pagina web è disponibile entro il timeout specificato se ne salva il contenuto nella base dati del sistema. Se la pagina Web non è disponibile tutte le operazioni successive non avranno effetto ed automaticamente il link verrà scartato dall'elenco dei risultati. Poiché il messaggio getPage() alla classe WebPage è bloccante e le operazioni di repository richiedono l'accesso alla base dati per salvare un flusso di dati (codice html),

il metodo repositoryLink() crea un nuovo thread per eseguire le proprie operazioni e rilascia immediatamente il controllo all'oggetto chiamante (accenniamo che l'oggetto chiamante appartiene alla classe Query).

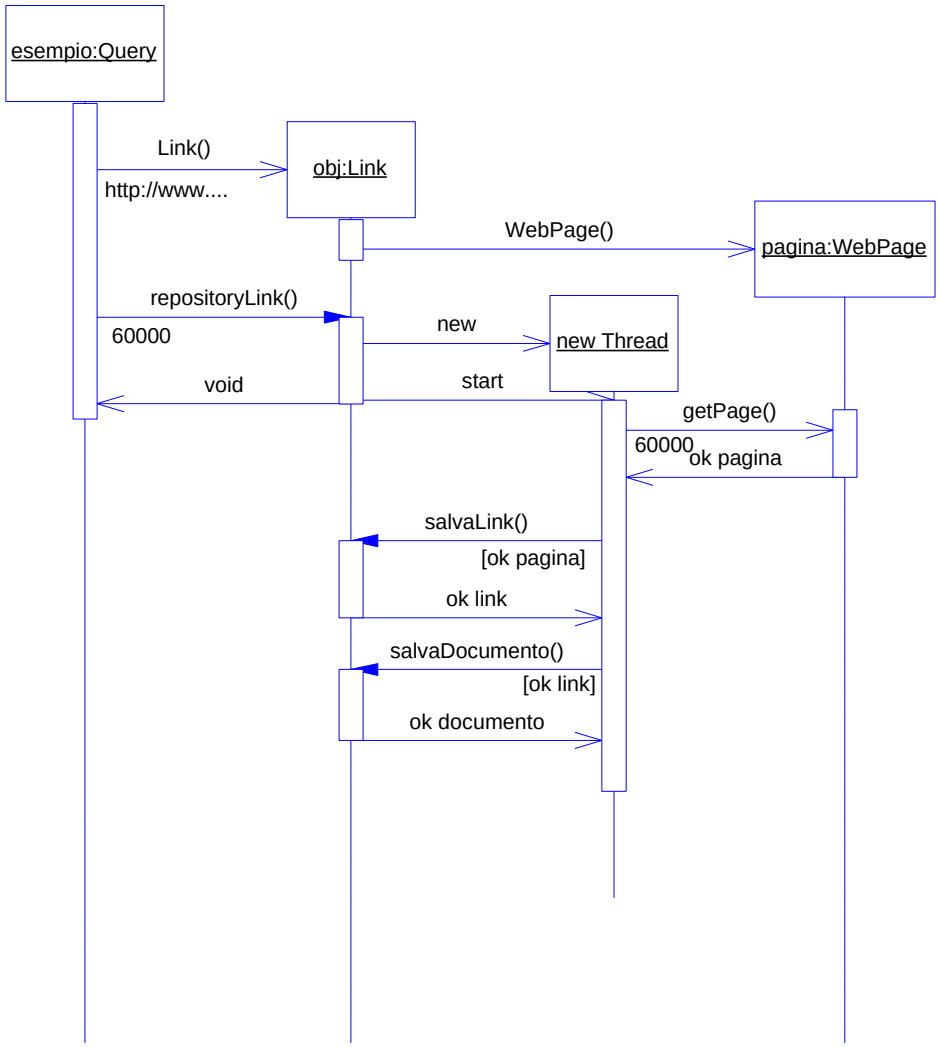

Figura 5.9: sequenza delle operazioni per il Repository

Dalla figura 5.9 si noti che nell'inviare il messaggio repositoryLink() viene passato il timeout, ovvero il tempo massimo di attesa per prelevare la pagina web.

Nel creare il link, viene assegnato un riferimento al motore di ricerca che lo ha restituito. Successivamente, per ogni motore di ricerca che individua il medesimo link, il gestore del sistema (oggetto Query) invia il messaggio addMotore() che consente di aggiungere l'oggetto MotoreDiRicerca al link.

L'operazione di repository builder si conclude quando sono stati individuati tutti i link e sono terminati tutti gli aggiornamenti. In questo caso si potrà inviare all'oggetto

link il messaggio salvataggioUltimo() che consente di calcolare la posizione media con il metodo calcolaPosizioneMedia() e di salvare i motori che hanno restituito il link, con il metodo salvalistaMotori().

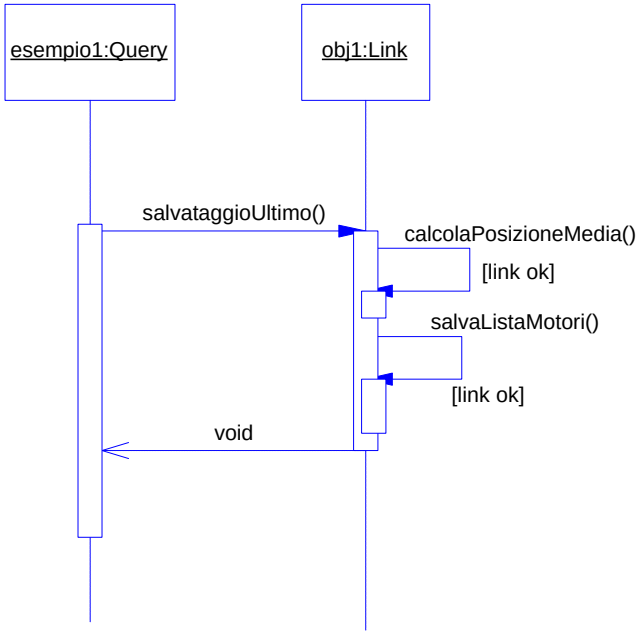

Figura 5.10: operazioni finali del repository builder.

Si noti che le operazioni possono non avere effetto se il link non è ok, ovvero se non è stata trovata la pagina oppure i dati non sono stati salvati nel sistema.

Ricapitolando, le operazioni di Repository Buider consistono in:

- ✓ salvare il link nella relazione Link;
- ✓ salvare il contenuto della pagina nella relazione Documenti;
- ✓ calcolare la posizione media e aggiornare la relativa tupla in link;
- ✓ salvare l'elenco dei motori associati nella relazione RisultatiMdR.

E' importate osservare che il calcolo della posizione media può essere effettuato solo alla fine poiché le operazioni di WebCatcher e le ricerche nei motori sono eseguite in maniera concorrente. Ciò sarà approfondito quando discuteremo della classe Query.

LA CLASSE MOTORIDIRICERCA

La classe motori di ricerca implementa il SEARCH ENGINE WRAPPER. Per creare un oggetto di tale classe è necessario prelevare le informazioni dalla base dati del sistema. Questo può essere fatto direttamente dal costruttore dell'oggetto oppure da oggetti esterni che, poi, possono passare i dati al costruttore. Quest'ultimo caso è efficiente quando bisogna costruire un set di oggetti MotoriDiRicerca.

All'oggetto sarà associato l'indirizzo http del relativo motore di ricerca. Inoltre, saranno elaborate le informazioni della tupla corrispondente nella base dati al fine di individuare:

- la rappresentazione della congiunzione;
- il modo in cui viene gestito il salto pagina nella restituzione dei risultati.

Tali operazioni vengono eseguite nella fase iniziale di definizione dell'oggetto.

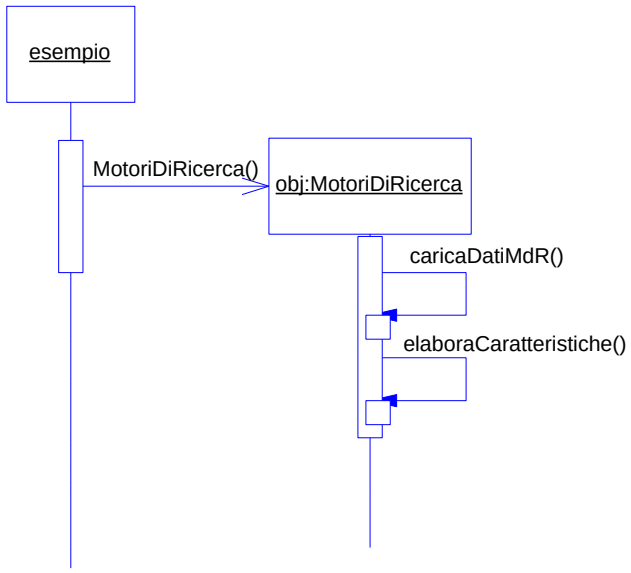

Figura 5.11: possibile creazione di un oggetto MotoriDiRicerca

Dopo aver definito il motore di ricerca sarà possibile passargli una interrogazione e l'oggetto stesso ne avvierà la ricerca. L'avvio della ricerca avviene con l'invio del messaggio avviaRicerca(), e sarà necessario fornire, come stringa, i vocaboli da ricercare e il numero massimo di link da prelevare. Il metodo avviaRicerca() invocherà il metodo costruisciUrl() che, implementa il QueryAdapter ed ha scopo di costruire un indirizzo http relativo al motore considerato e ad una determinata pagina dei risultati. Successivamente viene creato un oggetto WebPage con lo scopo di caricare la pagina relativa al colle-

gamento calcolato, funzionalità Web Catcher – metodo getPage(), e di prelevarne i link, funzionalità Parser - metodo estraiLinkAssolut() con filtro. L'operazione di ricerca si conclude quando il numero di link individuati si è esaurito o si è raggiunto il numero massimo di link impostato.

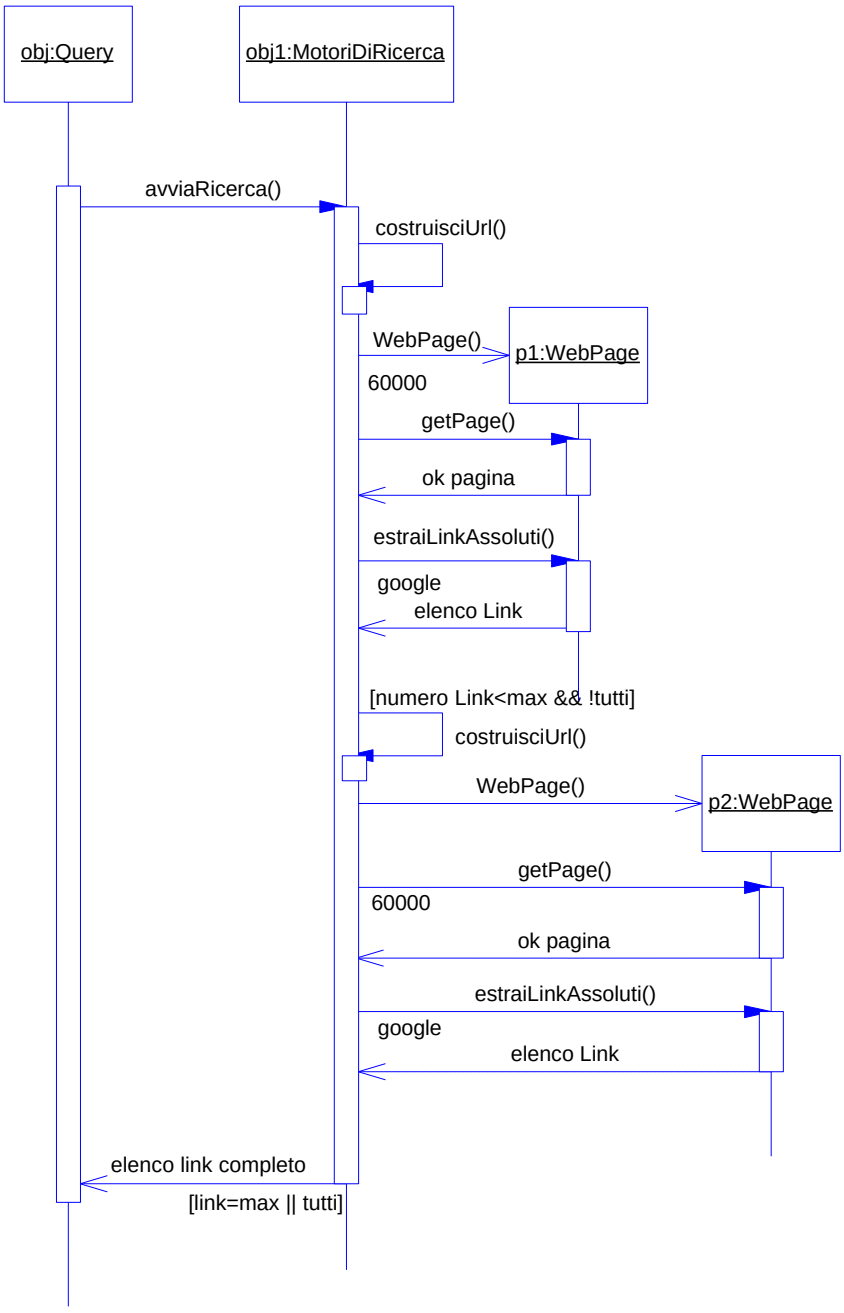

Figura 5.12: ecco come l'oggetto MotoreDiRicerca realizza il Search Wrapper.

L'elenco completo dei link, ordinato per posizione viene restituito all'oggetto chiamante.

LA CLASSE QUERY

E' la classe più importante dello spider: ha il compito di coordinare le attività di Search Engine Wrapper e di Web Spider.

Ogni oggetto di tale classe si riferisce ad una query precedentemente memorizzata nel database del sistema. Prima di poter invocare i messaggi di tale classe è necessario caricarne le informazioni dalla base dati. Tale operazione è svolta dal messaggio caricaDati(), che viene inviato verso l'oggetto Query con l'identificativo primario associato alla query salvata (idQuery). Il metodo provvederà a caricare dalla relazione Query la tupla indicata da idQuery, precisamente, le informazioni relative allo spider:

- i vocaboli da ricercare;
- il timeout impostato per le pagine web;
- il numero massimo di link per motore di ricerca.

Inoltre, caricherà dalla base dati le informazioni associate ai motori di ricerca selezionati, con i pesi relativi assegnati. Tutto questo è eseguito con un'unica interrogazione alla base dati. Per ogni motore di ricerca scelto viene creato un oggetto MotoreDiRicerca e lo si assocerà all'oggetto Query corrente.

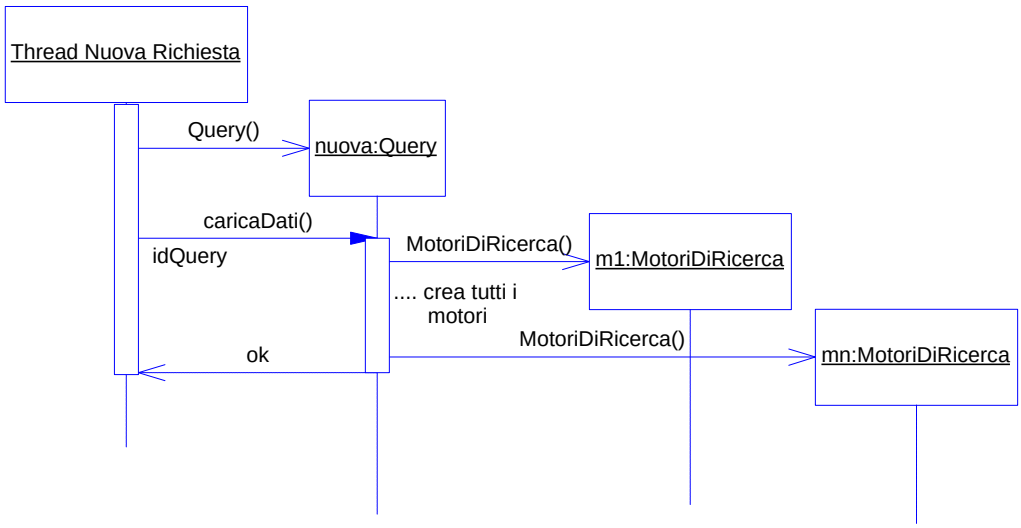

Figura 5.13: caricamento dei dati e creazione di un elenco di motori scelti.

Le operazioni di spidering vengono attivate mediante il messaggio avvia(), che deve essere invocato dopo caricaDati(). Tale messaggio è non bloccante: crea immediatamen-

te un nuovo Thread che effettua tutte le operazioni relative allo spidering, invocando avviaSpider().

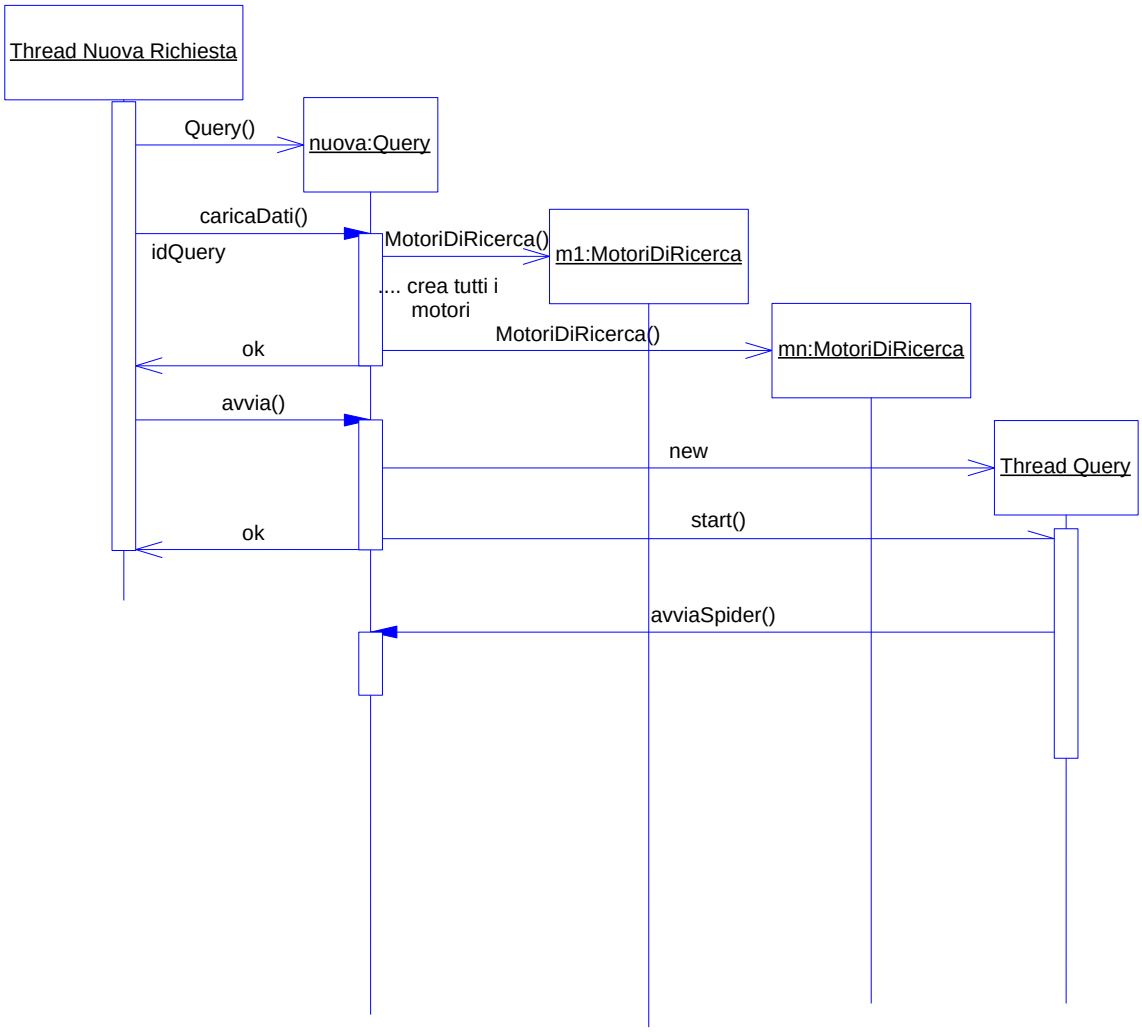

Figura 5.14: creazione di un Thread per eseguire la richiesta di spidering.

In questo modo quando l'interfaccia Server riceve una chiamata dal client, crea si un nuovo Thread (Thread Nuova Richiesta), come prevede il modello client-server, ma questo thread utilizza poi il metodo avvia() su un oggetto query, avviandone l'elaborazione e rilasciando immediatamente il client.

Esaminiamo ora la vera e propria attività di spidering, concentrando l'attenzione sul thread creato, Thread Query, e sull'oggetto nuova della classe Query.

Appena il Thread Query viene attivato invoca il messaggio avviaSpider(). Il server spider, per il suo accesso ad internet, nel contesto di tutto il sistema, è quello che ha i

tempi di elaborazione più alti: **l'azione eseguita dal metodo avviaSpider() è di fondamentale importanza per l'efficienza dello spider e, quindi, di tutto il sistema.**

Per avere uno spider performante è stato implementata la seguente strategia:

1. Creare una lista di oggetti Link ordinata per indirizzo http
2. Per l'i-esimo motore
   a. Avviare la ricerca nel motore e attendere l'elenco degli indirizzi http (stringhe), che rappresentano i link: vengono ordinati per posizione.
   b. Per il j-esimo indirizzo http:
      - controllare che esista nella lista (di cui al punto 1) un oggetto Link avente lo stesso indirizzo http;
      - se esiste, aggiungerne il motore di ricerca con la relativa posizione;
      - se non esiste creare un nuovo oggetto Link e inserirlo in posizione corretta all'interno della lista ordinata e avviarne il repository con il metodo repositoryLink(): viene avviato un nuovo thread.
   c. Eseguire le operazioni del punto b per ogni stringa http.
3. Eseguire le operazioni dal punto 2 per tutti i motori scelti
4. Attendere che venga ultimato il repository degli oggetti Link.
5. Inviare ad ogni oggetto Link che appartenga alla lista ordinata il messaggio salvataggioUltimo().
6. Memorizzare lo stato di **fine spidering,** salvaStatoElaborazione()**,** nell'entità stato della base dati, con le relative informazioni di log che avranno misurato i tempi di elaborazione.

Con l'operazione al punto 6 si conclude lo spidering per la query indicata e l'oggetto Query, con i relativi thread, cessa la propria attività.

Nella figura seguente indichiamo il diagramma di attività relativo alle operazioni di spidering. L'utilizzo delle corsie di marce (swimlanes) ci consente di indicare quale classe ha la responsabilità di una particolare azione.

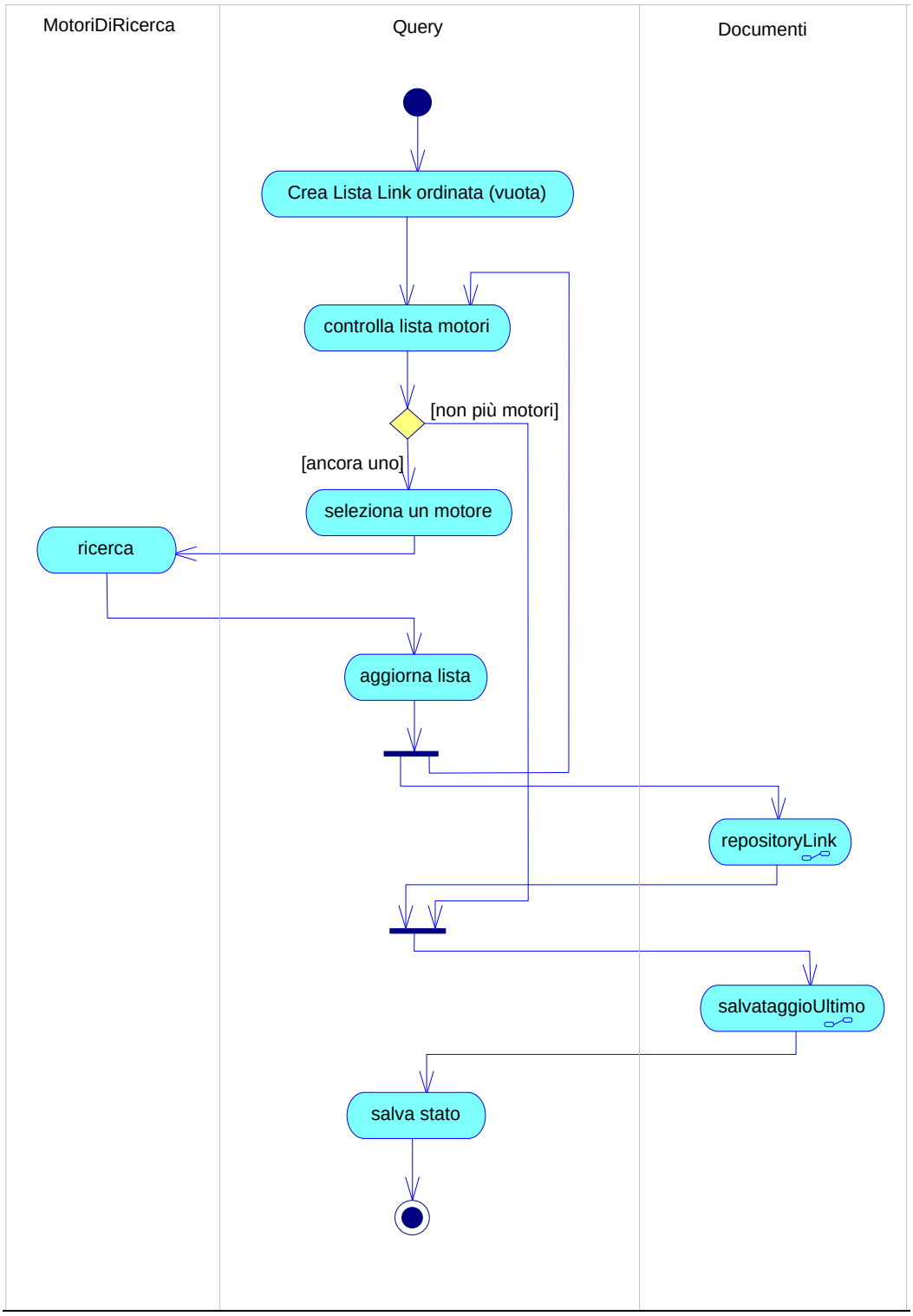

Figura 5.15: diagramma delle attività per il metodo avviaSpider().

Nella figura 5.15 si noti che l'avvio della ricerca nel corrispondente motore è effettuato, tranne la prima volta, in parallelo rispetto al repository dei link. L'attività composta repositoryLink sta ad indicare che per ogni nuovo link viene creato un thread che effettua il repository.

La sincronizzazione avviene quando sono esauriti i motori di ricerca selezionati per l'interrogazione e quando si è conclusa l'attività di repository.

Successivamente, per ogni oggetto Link, si effettuerà il salvataggioUltimo e alla fine l'oggetto query salva il proprio stato nel DB.

Ricordiamo che dopo aver inviato all'oggetto query il messaggio avvia(), per controllare quando l'attività di spidering è terminata è necessario accedere alla base dati e controllare nella relazione stato che ci sia una tupla relativa alla query data (idQuery) e al server spider (SS).

5.3 SERVER PREPROCESSOR

Il preprocessore delle pagine web ha il compito di eliminare tutto ciò che non ha a che fare con il contenuto informativo della pagina e di predisporre il documento per l'analisi semantica.

Il Server si riferisce alle entità Documenti e DocumentiPreProcessati della base dati. Ha il compito di prelevare dall'entità Documenti nuove pagine Web, ossia quelle appena inserite dalla spider, di elaborarle, individuando il contenuto informativo del titolo, della descrizione, delle keyWords e del corpo, e di salvarne i risultati nell'entità DocumentiPreProcessati. Il preprocessore riesce a distinguere nell'entità Documenti le pagine Web nuove da quelle già elaborate grazie all'indicatore di stato del documento (statoPreProcessor): quando il documento è nuovo il valore dell'attributo è zero, appena viene elaborato diventa uno.

Al package ServerPreprocessor appartengono le seguenti classi:

- Documento:

  e' la classe che fa riferimento alla pagina web depositata dallo spider nel sistema: ne effettua l'elaborazione;

- PreProcessa:

  rappresenta un thread, appartenente ad un gruppo, che ha il compito di gestire l'elaborazione di un elenco di nuovi documenti web prelevati dalla base dati;

- QueryPrePro:

  È la classe che fa riferimento alla query di cui si deve fare il preprocessing.

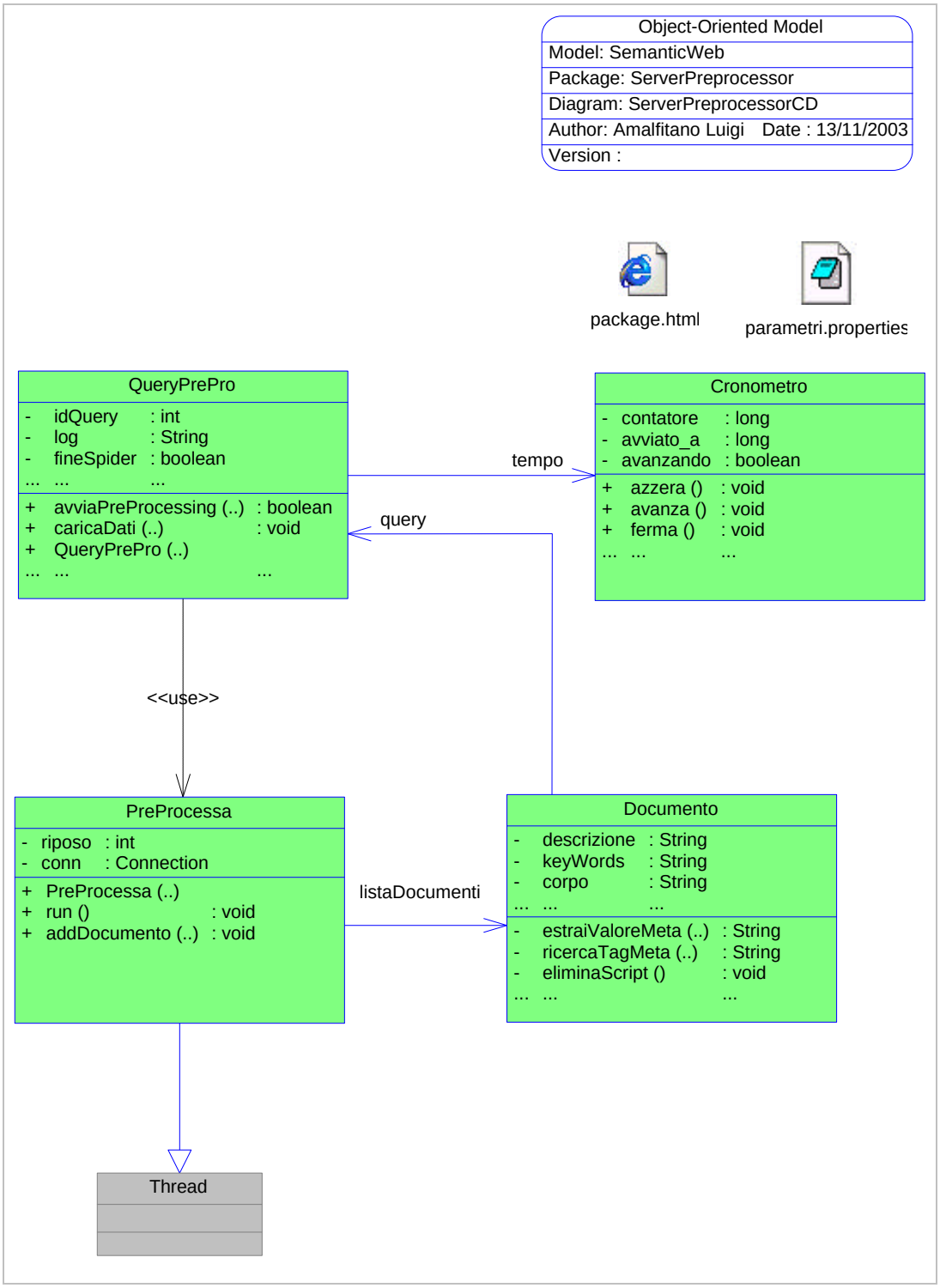

Figura 5.16: diagramma delle classi relativo al Server Preprocessor

La classe Preprocessa risulta associata con la classe Documento: questo perché ha il compito di gestire l'elaborazione di un set di documenti (listaDocumenti).

La classe Documento è associata alla classe QueryPrePro: ogni documento appartiene ad una data query, in questo caso quella di cui si è richiesta l'elaborazione di preprocessing (query).

L'associazione della classe Query alla classe Cronometro ha il compito di determinare le informazioni di log (tempi di elaborazione) relativi al preprocessing di una determinata query.

La classe Query utilizza un insieme variabile di oggetti Preprocessa: uno per ogni nuovo accesso alla base dati che restituisce un elenco di documenti da preprocessare.

Il file package.html server per la documentazione e il file parametri.properties permette di impostare i parametri del server.

LA CLASSE DOCUMENTO

La classe Documento rappresenta il vero e proprio DOCUMENT PREPROCESSOR esposto nel capitolo 3. Gli oggetti appartenenti a tale classe hanno il compito di elaborare il documento html. Ogni oggetto è associato ad una pagina web per la quale si effettuano, nell'ordine indicato, le seguenti operazioni:

1. eliminare gli script lato client e i commenti html;
2. eliminare oggetti speciali (tipo macromedia flash);
3. eliminare le applet;
4. individuare e isolare le parti essenziali di un documento html, ovvero <head> e <body>;
5. individuare e isolare, se presente, il titolo;
6. individuare e isolare, se presente, la descrizione;
7. individuare e isolare, se presente, l'elenco delle parole chiavi;
8. eliminare dal corpo del documento html (parte body) tutti i tag html;

9. sottoporre il corpo del documento html (parte body) ad operazione di filtraggio per sostituire i caratteri speciali e eliminare gli spazi vuoti.

Alla fine di tali operazioni sarà stato individuato:

- ✓ il contenuto informativo del titolo;
- ✓ il contenuto informativo della descrizione;
- ✓ l'elenco delle parole chiavi;
- ✓ il contenuto informativo relativo al corpo del documento.

Le informazioni così elaborate verranno salvate nel database del sistema e rese disponibili per operazioni successive.

Per implementare tutte le operazioni elencate è stato necessario prestare la massima attenzione all'efficienza: diventa fondamentale quando l'oggetto si riferisce ad una pagina Web molto grande. L'ordine elencato per le operazioni garantisce efficacia, robustezza ed efficienza: vengono realmente individuate in maniera molto veloce (non più di 5 sec) le informazioni fondamentali del documento, anche nei caso di utilizzo non corretto dei tag html (per esempio nel caso in cui non si utilizzano i tag head e body).

Poiché le pagine web non hanno una rigida e ben definita struttura è stato necessario effettuare su tutto il documento la ricerca e l'eliminazione degli script: in molti casi si trovano in punti diversi della pagina e possono contenere anche tag html che non vengono visualizzati inizialmente, ma che non apportano contenuto informativo.

La creazione di un oggetto Documento prevede l'assegnazione di tutto il contenuto della pagina web, oltre all'indirizzo http associato; successivamente ogni oggetto dovrà:

- ✓ elaborare il documento;
- ✓ salvare le informazioni individuate nella base dati;

Ciò è possibile inviando all'oggetto il messaggio settaCampi(). Tale metodo invocherà i seguenti metodi privati:

1. eliminaScript(), per eliminare gli script lato client;
2. eliminaCommentiHtml(), per eliminare tutti i commenti html;

3. setHead(), per individuare ed associare all'oggetto la parte head;
4. setBody(), per individuare ed associare all'oggetto la parte body;
5. setTitolo(), per individuare ed associare all'oggetto il titolo;
6. setDescrizione(), per individuare ed associare all'oggetto la descrizione;
7. setKeyWords(), per individuare ed associare all'oggetto l'elenco delle parole chiavi del documento;

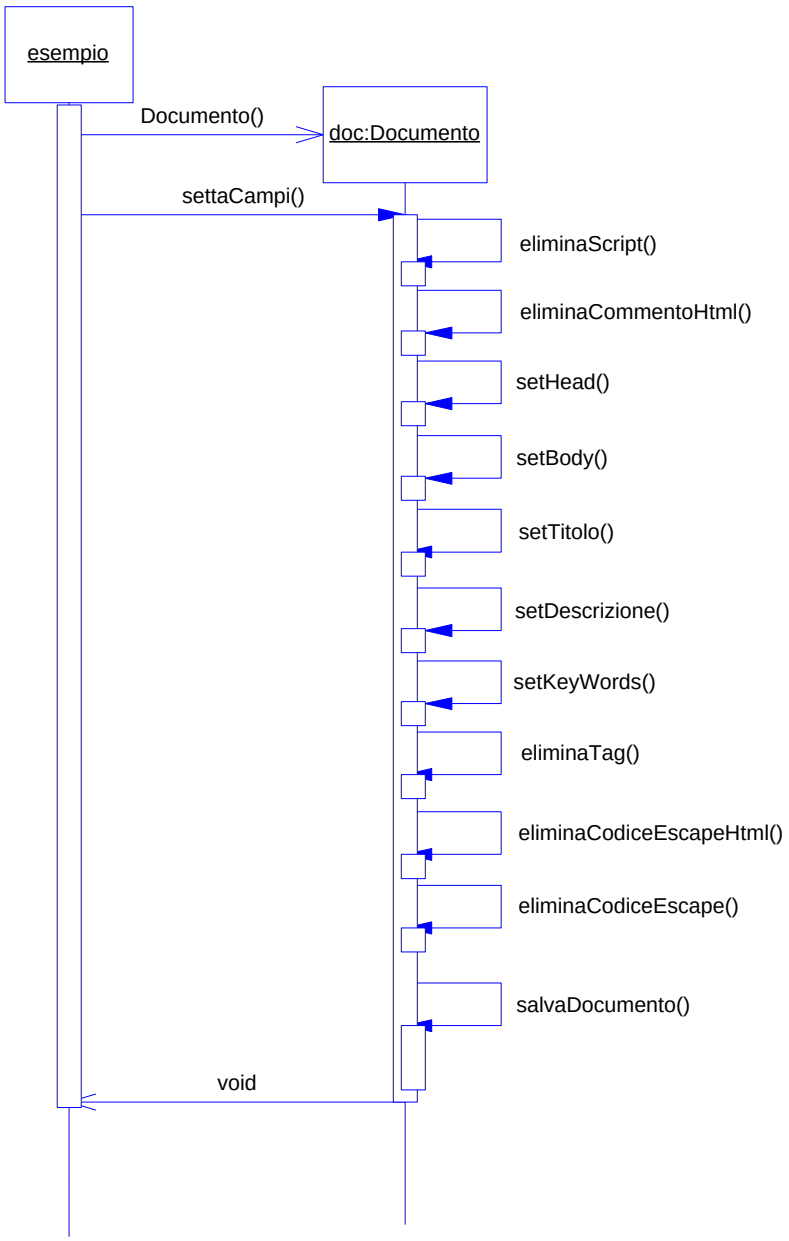

Figura 5.17: elaborazione del documento html.

Successivamente alla parte body del documento saranno applicati i seguenti metodi privati:

1. eliminaTag(), per eliminare tutti i tag html;
2. elimaCodiciEscaperHtml(), per eliminare tutti i codici html, scritti o meno in formato unicode: &#xxx; per unicode ed &nome; per html;
3. eliminaCodiciEscape(), per eliminare tutti i codici escare: tabulazioni (\t), fine riga (\r) e fine riga e ritorno a capo (\n);

Naturalmente, i metodi del tipo elimina...(), sostituiranno il codice eliminato con uno spazio, e il metodo eliminaSpazi() garantisce che tutto il documento e le sue parti, non abbiano spazi consecutivi.

LA CLASSE PREPROCESSA

La classe PreProcessa ha il compito di gestire l'elaborazione di un insieme di documenti. Ogni oggetto di tale classe rappresenta un nuovo Thread, appartenente ad un gruppo e creato da un oggetto della classe query. All'atto della creazione dell'oggetto sarà necessario assegnare il nome del Thread e del gruppo di appartenenza. Inoltre, sempre all'atto della creazione viene definito un elenco vuoto che, in una fase successiva verrà riempito da oggetti documento.

Per utilizzare l'oggetto della classe PreProcessa sarà quindi necessario:

1. crearlo, definendo le caratteristiche del thread;
2. con il metodo addDocumento() assegnare un documento all'oggetto PreProcessa;
3. dopo aver assegnato tutti gli oggetti avviare il thread con il metodo start();

Tramite il messaggio addDocumento è possibile inviare all'oggetto PreProcessa il contenuto completo della pagina Web (come stringa) ed il link associato. Sarà poi il metodo addDocumento() che si preoccuperà di creare un nuovo oggetto della classe Documento e di inserirlo nell'elencoDocumenti. Il metodo addDocumento() dovrà essere

utilizzato per ogni nuova pagina web trovata dallo spider ed inserita nella base dati del sistema.

Dopo aver riempito l'elenco dei documenti è possibile avviare il thread PreProcessa con il metodo start(): verrà avviato il preprocessing sui documenti caricati. I documenti verranno processati in maniera sequenziale.

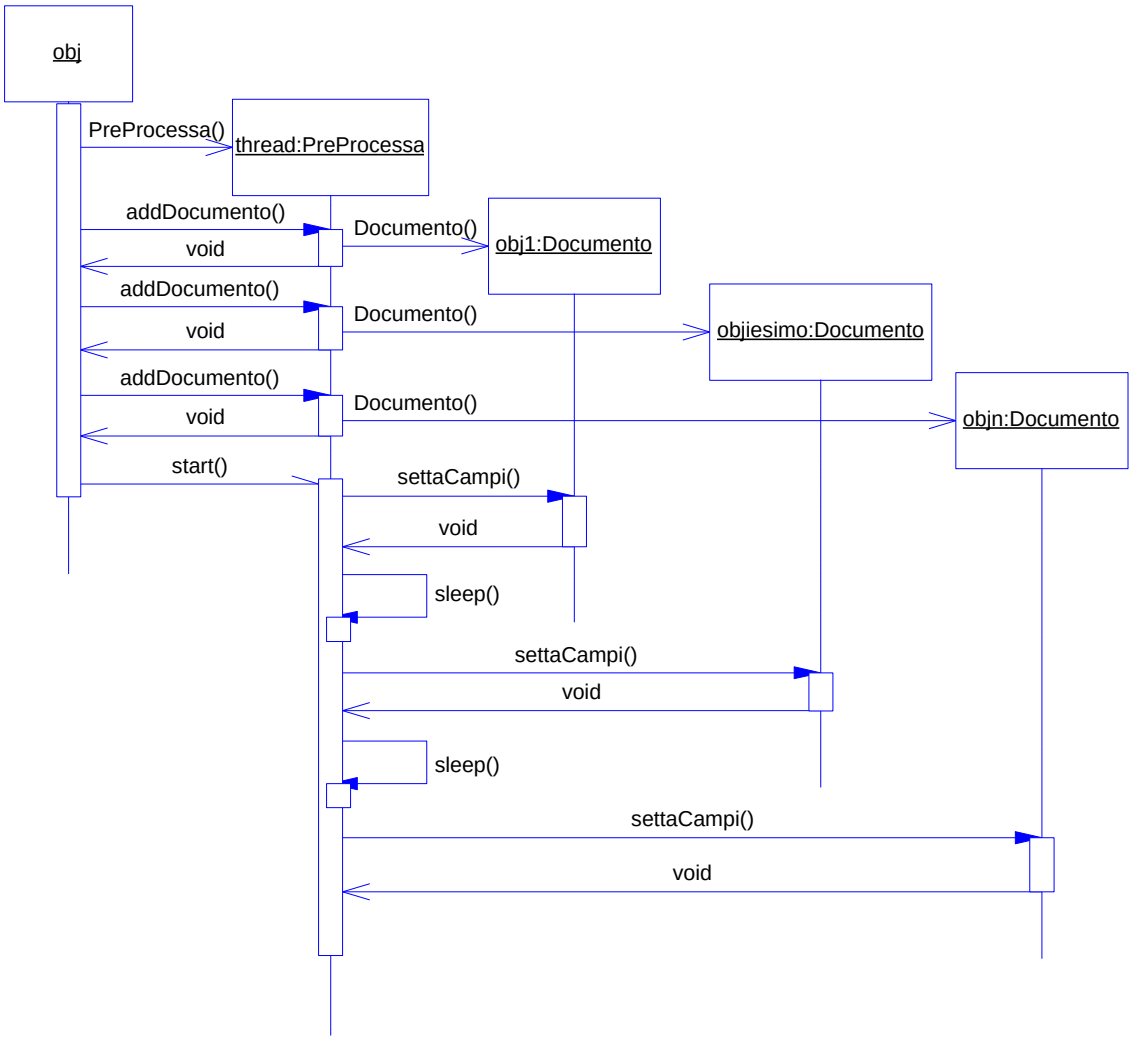

Figura 5.18: schema sequenziale di un oggetto della classe PreProcessa.

Tra l'elaborazione di due documenti è possibile sospendere per alcuni millisecondi il thread corrente per dare spazio ad altri thread PreProcessa.

Il Thread terminerà il suo lavoro quando saranno state preprocessate tutte le pagine html. Alla fine, quindi, la base dati conterrà nella relazione DocumentiPreProcessati i documenti elaborati dal Thread.

LA CLASSE QUERYPREPRO

La classe QueryPrePro ha lo scopo di avviare il preprocessing per una data interrogazione e di gestire l'accesso a nuovi documenti da preprocessare.

Per eseguire tali compiti un oggetto della classe ha solo bisogno dell'identificatore della query: non è richiesto all'utente alcun parametro di configurazione per il preprocessing, a differenza dello spider per il quale erano richieste diverse informazioni. L'identificatore della query, assegnato in fase di costruzione dell'oggetto, consente di accedere direttamente ai documenti scaricati dallo spider senza necessità di reperire altre informazioni dalla base dati. Naturalmente, è richiesto che l'identificatore si riferisca realmente ad una query presente nella base dati del sistema.

Le operazioni di preprocessing vengono attivate mediante il messaggio avviaPreProcessing(), che può essere invocato appena dopo aver definito l'oggetto. Tale messaggio è non bloccante: crea immediatamente un nuovo Thread che effettua tutte le operazioni relative al preprocessing dei documenti.

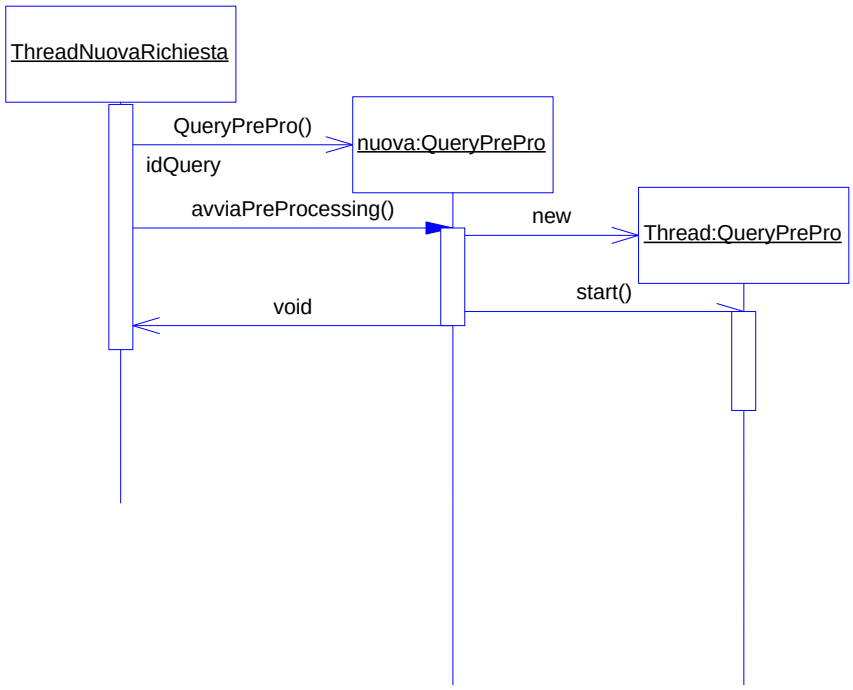

Figura 5.19: avvio non bloccante delle operazioni di preprocessing.

In questo modo quando l'interfaccia Server riceve una chiamata dal client, crea si un nuovo Thread (Thread Nuova Richiesta), come prevede il modello client-server, ma que-

sto thread utilizza poi il metodo avviaPreProcessing() su un oggetto QueryPrePro, avviandone l'elaborazione e rilasciando immediatamente il client.

Il cuore della gestione del preprocessing risiede nel Thread creato mediante il messaggio avviaPreProcessing: il Thread deve caricare dalla base dati nuovi documenti fino a quando:

1. il server spider non termina la propria elaborazione (per la query data);
2. non ci sono più pagine web da prelevare;

Tutto ciò viene realizzato all'interno di un ciclo che verrà eseguito fino a quando lo stato di elaborazione dello spider è falso, ovvero fino a quando è in attività. Il ciclo comprende le seguenti operazioni:

a. viene settato lo stato dello Spider, mediante il metodo setFineSpider();
b. vengono prelevate eventuali nuove pagine web dalla base dati, mediante il metodo caricaDati();
c. si controlla lo stato di elaborazione dello spider, con il metodo getFineSpider().

L'operazione principale del ciclo è rappresentata dal metodo caricaDati(). Tale metodo ha accesso alla base dati ed effettua nell'ordine indicato le seguenti operazioni:

- blocco write per la tabella Documenti;
- preleva tutti i nuovi documenti inseriti dallo spider (ossia quelli che per l'attributo **statoPreProcessor** assumono valore falso);
- aggiorna lo stato dei documenti prelevati, assegnando per l'attributo statoPreProcessor valore vero;
- sblocca la tabella documenti;
- se ci sono nuovi documenti:
  - crea un thread PreProcessa;
  - per ogni nuovo documento invoca il messaggio addDocumento();
  - avvia il thread PreProcessa;

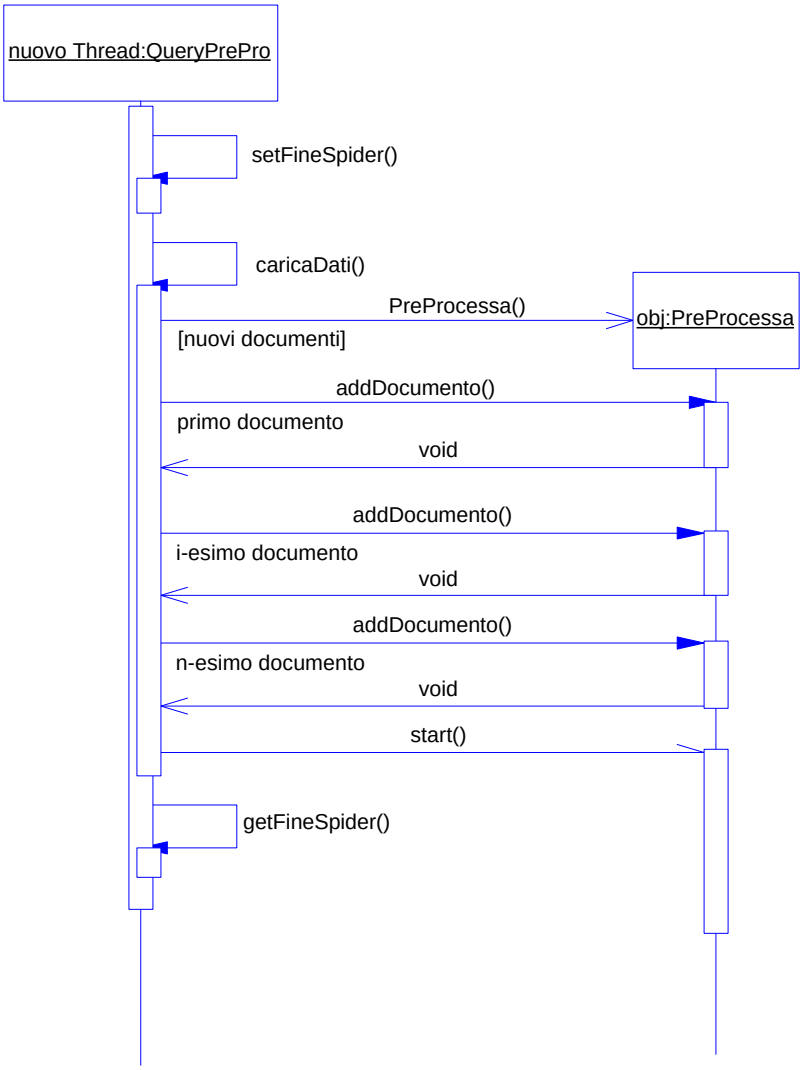

Figura 5.20: avvio del preprocessing per un elenco di nuovi documenti.

Il metodo setFineSpider() ha accesso alla base dati del sistema per controllare, nell'entità stato, l'attività del server spider per la query di cui si sta effettuando il preprocessing: se l'attività è terminata l'indicatore di stato associato assumerà valore vero, altrimenti assumerà valore falso.

Il metodo getFineSpider() ha il compito di leggere il valore assegnato all'indicatore di stato relativo allo spider.

E' importante notare l'ordine delle operazioni all'interno del ciclo: verifica le condizioni 1 e 2 di esecuzione del Thread.

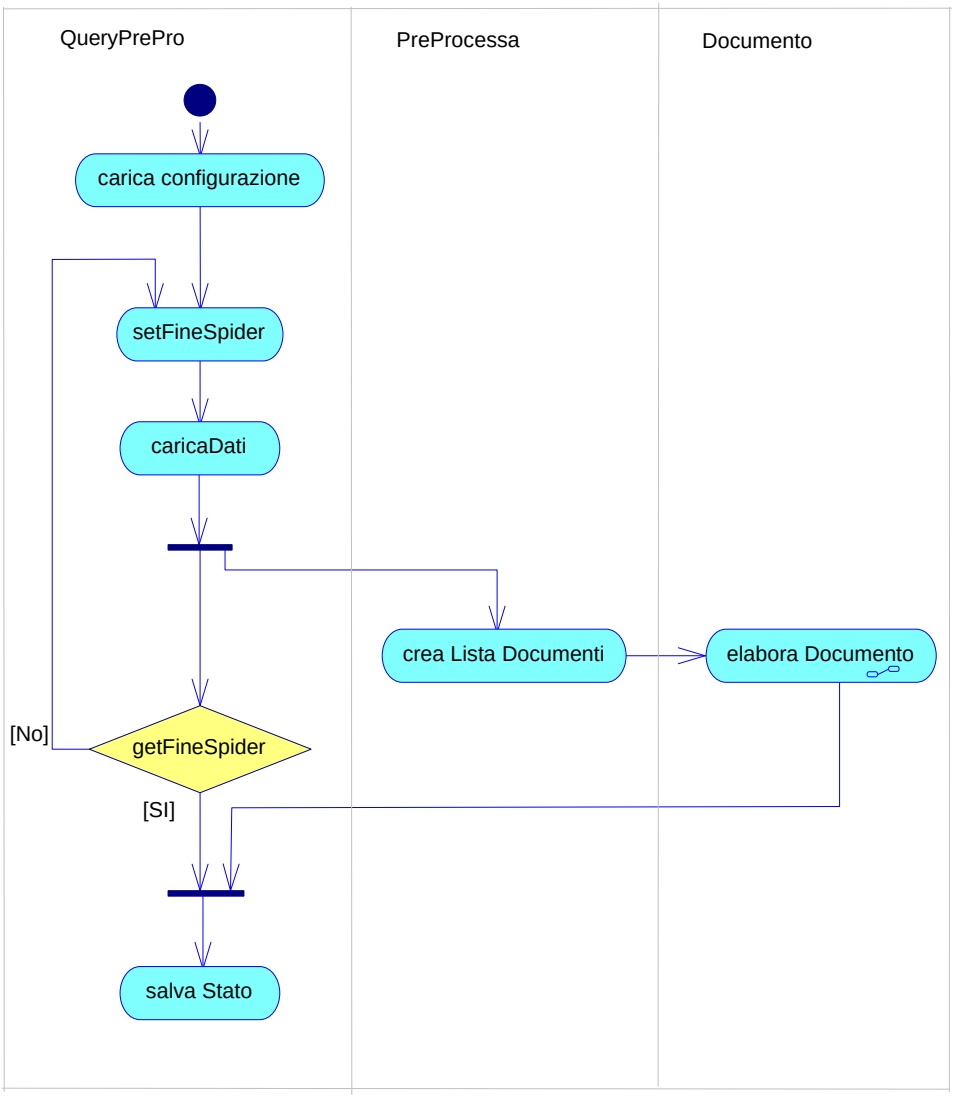

Figura 5.20: attività di gestione delle operazioni di preprocessing.

Quando lo spider ha terminato la propria elaborazione e sono state prelevate tutte le pagine web dalla base dati, è necessario attendere che i vari thread creati dal metodo caricaDati(), in chiamate successive, abbiano cessato la propria attività; ciò indica il fatto che sono stati preprocessati tutti i documenti presenti nella base dati. L'attività di preprocessing termina, quindi, invocando il messaggio salvaStatoElaborazione(). Tale metodo consente di salvare nella tabella stato della base di dati l'informazione di fine attività del server Preprocessor (SP) per la query data e le informazioni di log relative ai tempi di elaborazione.

GESTIONE DEI THREAD PREPROCESSA E TEMPIFICAZIONE

Da quanto detto nei paragrafi precedenti, si può constatare che l'attività di preprocessing dei documenti appartenenti ad una data query è condotta da un insieme di thread che vengono creati ogni volta che si accede alla base dati e si individuano nuovi documenti.

Per un funzionamento efficiente del sistema sarebbe necessario che i documenti prelevati dalla base dati per l'elaborazione siano distribuiti quanto più uniformemente possibile tra i vari thread che possono essere creati. Tuttavia, la distribuzione dei documenti fra i vari thread non è molto semplice poiché essi vengono inseriti nel sistema in tempi successivi, che dipendono molto dall'attività di spidering.

Si è pensato, quindi, di stabilire una tempificazione sul metodo caricaDati(): esso verrà invocato con una certa frequenza che è possibile stabilire e impostare in maniera dinamica. Si definisce come tempo di caricamento fra i dati (tcd) l'intervallo fra due chiamate al metodo caricaDati(). Tale valore è espresso in millisecondi e viene impostato in un file di proprietà.

Dopo aver avviato tutti i thread, quello principale, di tipo QueryPrePro, si pone in attesa della loro terminazione: con una certa frequenza è possibile controllarne il loro stato di terminazione. L'intervallo, in millisecondi, fra due controlli successivi sullo stato dei thread viene definito come tempo fra thread (tft) e può essere impostato in maniera dinamica.

I due valori, tcd e tfd, vengono definiti nel file di proprietà (parametri.properties) presente nel package e vengono caricati ogni volta che si invia il messaggio caricaDati(), come indicato in figura 5.20. Modificando tali valori è possibile incidere in maniera notevole sull'efficienza del Server Preprocessor. L'impostazione corretta di tali valori è legata alla attività del Server Spider e alle caratteristiche hardware a disposizione del Server Preprocessor.

Una possibile evoluzione del sistema potrebbe prevedere un controllo a ciclo chiuso, che, analizzando i tempi di elaborazione, possa intervenire sui due parametri al fine di garantire un valore minimo di efficienza.

5.4 SEMANTIC NETWORK

L'attività di mining è condotta basandosi su un insieme di classi, opportunamente collegate e rappresentanti le informazioni presenti all'interno del Semantic Knowledge Base della base dati. Tale struttura prende il nome di Semantic Network e rappresenta un package a disposizione del Server Miner.

La Semantic Network ha il compito di caricare in memoria e gestire l'ontologia: viene creata una "rete" di istanze di oggetti, una per ogni dominio.

Il class Diagramm è composto dalle seguenti classi:

- Dominio:

  ogni oggetto di tale classe individua un dominio presente all'interno della SKB;

- Concetto:

  rappresenta i concetti presenti nella SKB;

- Legame:

  rappresenta le distanze probabilistiche fra due concetti appartenenti allo stesso dominio;

- Parola:

  ogni oggetto rappresenta una parola di un determinato dominio;

- StopWords:

  è una classe statica che ha il compito di eliminare dal testo tutto ciò che è irrilevante per l'analisi semantica;

- Testo:

  ogni oggetto della classe rappresenta un testo che può essere analizzato semanticamente.

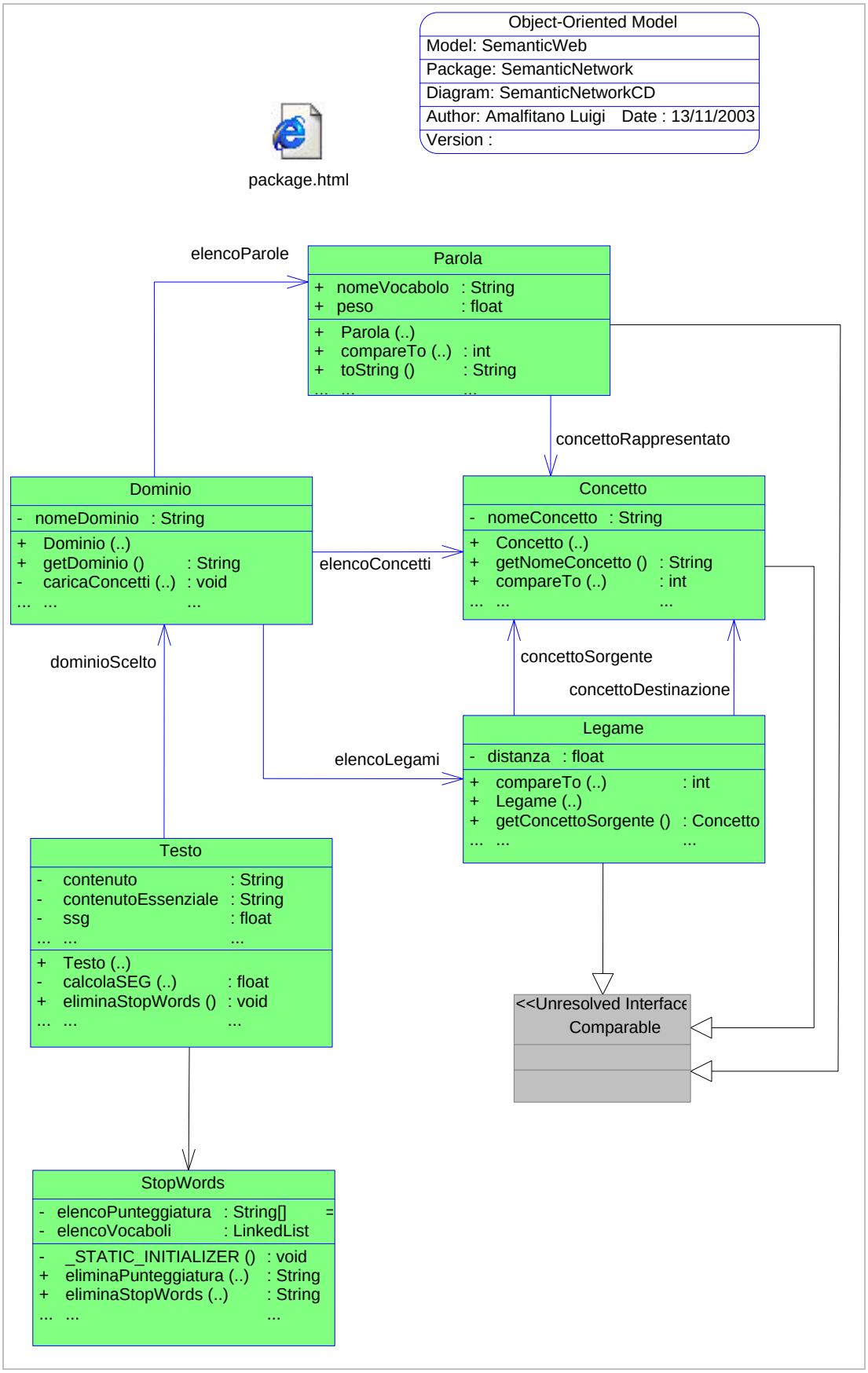

Figura 5.21: diagramma delle classe della Semantic Network.

Aver organizzato la SKB in un package a parte ci consente di isolare tutte le informazioni sull'ontologia in un unico punto del sistema. In questo modo, nel caso in cui si volesse modificare in maniera radicale l'ontologia, oppure modificare la metrica, sarà necessario intervenire solo nella Semantic Network.

Dalla figura 5.21 si evince il modo in cui le varie classi interagiscono fra loro (il file package.html è necessario per la documentazione).

E' importante osservare che il perno centrale è rappresentato dalla classe Domino. Essa è associata con le Parole, con i Concetti e con i Legami.

L'associazione con i concetti sta ad indicare che ogni concetto appartiene ad un determinato dominio: l'oggetto dominio all'atto della creazione accederà alla base dati per individuare tutti i propri concetti e inserirli in un elenco ordinato per nome.

L'associazione tra la classe Concetti e la classe Legame rappresenta l'informazione sulla distanza probabilistica fra concetti. Sarà sempre la classe Dominio che gestirà i legami: ogni oggetto della classe Dominio accede alla base dati per prelevare le informazioni relative alle distanze probabilistiche fra i propri concetti e creerà un elenco di legami, ordinati per concetto sorgente e concetto destinazione. Ogni legame sarà caratterizzato da due istanze, una per l'oggetto concetto sorgente, l'altra per l'oggetto concetto destinazione; prima di definire l'oggetto Legame è necessario definire gli oggetti Concetto.

La classe Parola risulta associata alla classe Concetto. Ciò indica che ogni vocabolo presente all'interno della SKB può essere utilizzato per definire una parola, ossia essere associato con un peso ad un concetto appartenente ad un determinato dominio.

E' importante osservare che nel momento in cui si vuole effettuare una modifica radicale della SKB è necessario intervenire solo ed esclusivamente in questo package.

DEFINIZIONE DELLA SEMANTIC NETWORK

Per definire la Semantic Network è necessario creare gli oggetti Dominio. All'atto della creazione, l'oggetto dominio accederà direttamente alla base dati prelevando tutte le informazioni che lo interessano: concetti, legami, vocaboli.

Il metodo privato caricaConcetti() consente all'oggetto Dominio di creare oggetti Concetto e di inserirli in un elenco ordinato, elencoConcetti. Successivamente verrà in-

vocato il metodo privato caricaLegami(), che carica dalla base dati i legami fra concetti e crea gli oggetti Legame: al costruttore dell'oggetto vengono passate le relative istanze degli oggetti Concetto presenti in elencoConcetto. Gli oggetti legame verranno inseriti in un elenco ordinato chiamato elencoLegami. In ultimo viene invocato il metodo privato caricaParole(), per accedere alla base dati e caricare le parole associate al dominio. Per ogni parola viene creato il relativo oggetto passando come concetto una istanza presente all'interno di elencoConcetti; le parole caricate vengono inserite in un elenco ordinato chiamanta elencoParole.

Nella definizione dei vari elenchi ordinati si utilizzano strutture ad accesso casuale, ciò per utilizzare l'algoritmo di ricerca binaria.

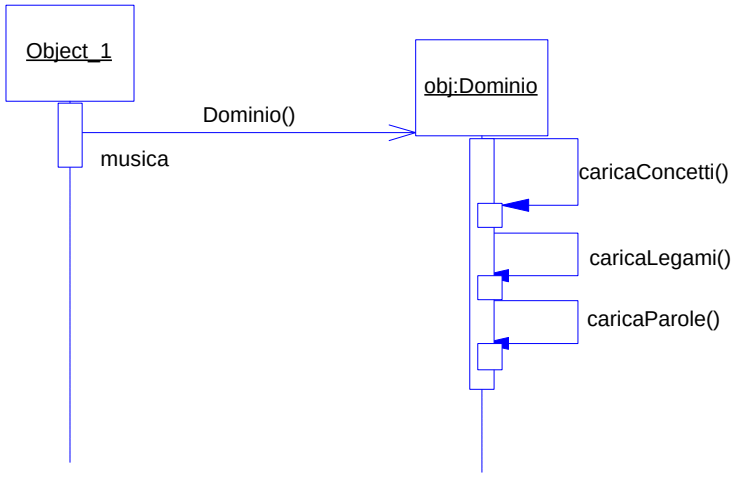

Figura 5.22: operazioni eseguite dal costruttore del Dominio.

Dopo aver definito un dominio sarà possibile controllare se un vocabolo, indicato come stringa, è associato al dominio e in che modo. Di seguito elenchiamo i metodi pubblici che possono essere utilizzati

- getPesoParola():

  restituisce il peso del vocabolo passato al messaggio, se associato al dominio come parola, zero altrimenti.

- getConcettoRappresentato():

  individua il concetto rappresentato dal vocabolo passato, se presente, **null** se non esiste.

- getDistanzaConcettiRappresentati():

individua la distanza probabilistica tra i due concetti indicati nel messaggio. E' possibile utilizzare il messaggio anche passando due vocaboli: vengono dapprima ricercate le parole, poi i concetti e alla fine il legame.

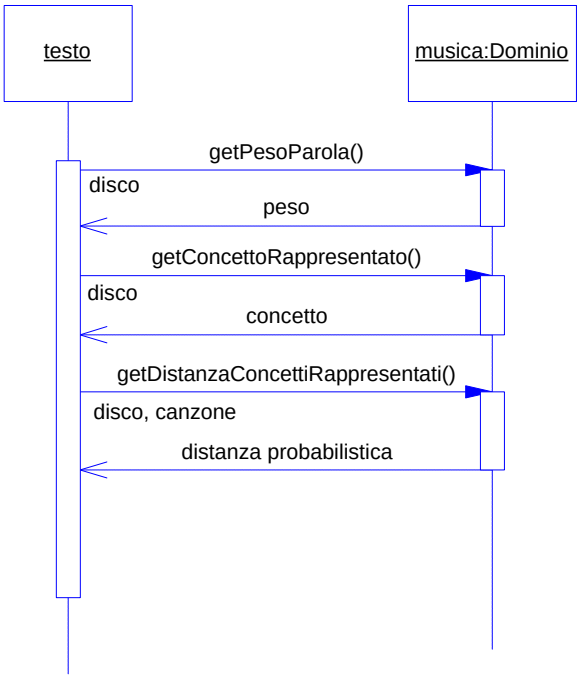

Figura 5.23: probabile utilizzo del dominio musica.

Utilizzando esclusivamente tali metodi sarà possibile, come vedremo analizzando la classe Testo, effettuare l'analisi semantica.

LA CLASSE STOPWORDS

La classe StopWords è di tipo statico ed ha un set di messaggi che consente di elaborare un determinato testo: il testo è passato al metodo come parametro e in uscita il metodo restituisce il testo elaborato.

Nella fase di inizializzazione della classe viene caricato dalla base dati del sistema, dalla entità StopWordList, l'elenco delle parole non attinenti ad alcuna analisi semantica. A tale elenco si aggiunge poi quello rappresentante i caratteri di punteggiatura; inoltre vengono considerate parole non attinenti tutte quelle composte da tre caratteri. Da ciò si comprende che nell'entità non risulta necessario inserire occorrenze rappresentanti i segni di punteggiatura e parole brevi: occorre inserire parole con più di tre caratteri.

La classe StopWords prevede una serie di metodi statici, ciascuno con un compito ben preciso. Il messaggio più importante è eliminaStopWords(). Tale metodo elimina dalla stringa passata in ingresso tutti gli elementi che non sono attinenti all'analisi semantica, sostituendoli con uno spazio. Il metodo eliminaStopWords(), prima di eliminare dalla stringa i vocaboli appartenenti alla StopWordList, invoca i seguenti metodi:

- ✓ eliminaPunteggiatura(), per eliminare i segni di punteggiatura
- ✓ eliminaParoleBrevi(), per eliminare le parole con meno di tre caratteri;

In questo modo si ottimizza il tempo di elaborazione per la preparazione del testo all'analisi semantica.

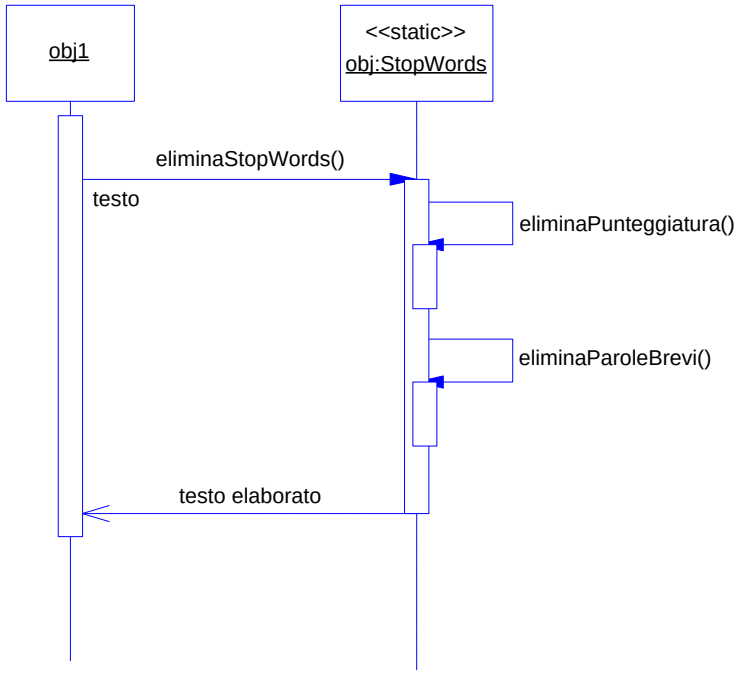

Figura 5.24: utilizzo dei metodi statici della classe StopWords.

La gestione dei vocaboli non attinenti all'analisi semantica non dipende dal dominio scelto, per cui è sempre possibile applicare tali metodi.

Il testo da elaborare è sempre rappresentato come stringa. In uscita si ha una copia della stringa in ingresso, a meno delle stop word individuate.

LA CLASSE TESTO

Gli oggetti di tale classe rappresentano un testo da analizzare semanticamente. La costruzione dell'oggetto prevede l'assegnazione del testo da analizzare e l'attribuzione di un oggetto della classe dominio; successivamente sarà possibile analizzare il testo in riferimento al dominio associato. L'attivazione dell'elaborazione semantica è molto semplice, basta invocare i seguenti messaggi:

- ✓ eliminaStopWords(), per eliminare i vocaboli non attinenti;
- ✓ assegnaVotoSemantico(), per assegnare al testo il voto SSG e SeG;

Il metodo assegnaVotoSemantico() ha accesso all'oggetto Dominio: viene utilizzato per individuare le parole e i concetti. Queste informazioni, come visto nel capitolo 2, sono alla base della metrica utilizzata.

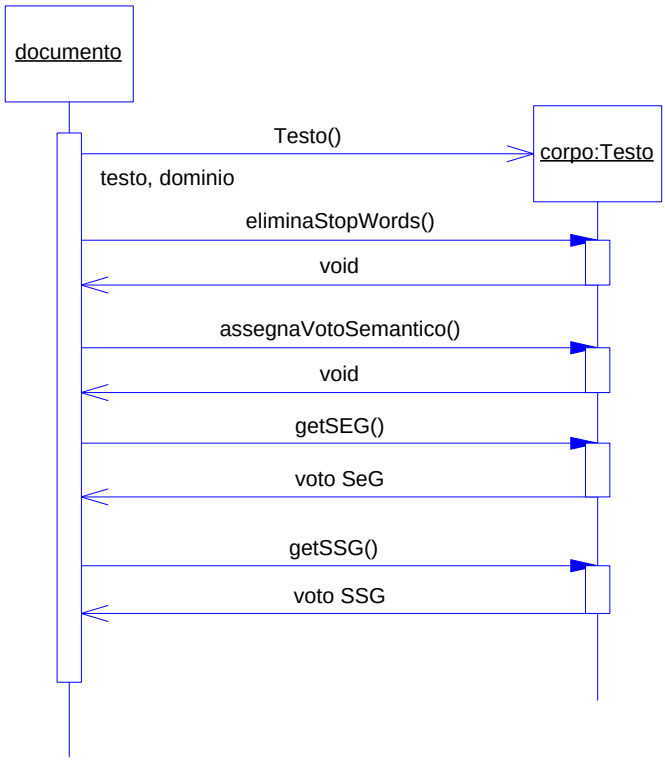

Figura 5.25: utilizzo di un oggetto della classe Testo.

Si noti nella figura 5.25 i metodi get: il metodo getSEG() consente di leggere il voto semantico, il metodo getSSG() consente di leggere il voto semantico-sintattico.

5.5 SERVER MINER

Il Server Miner implementa il Miner del sistema. Esso fa riferimento all'entità DocumentiPreProcessati e all'entità Link della base dati: preleva i documenti preprocessati, ne effettua l'analisi semantica, calcolandone il voto ed aggiornando il relativo link. Dopo aver elaborato tutti i documenti preprocessati viene determinato il voto finale, che sarà insierito nella entità link. L'operazione di mining è concorrente alle operazioni di spidering e di preprocessing. Il miner preleva i documenti preprocessati mano a mano che vengono elaborati dal Server Preprocessor, riuscendo a distinguere i nuovi documenti da elaborare grazie ad un indicatore di stato, statoMiner, presente nell'entità DocumentiPreProcessati. Il valore di statoMiner sarà zero se il documento non è stato ancora elaborato dal miner, uno se è stato elaborato.

Poiché il mining è concorrente alle operazione degli altri server, potrebbe accadere che il documento sia elaborato prima che venga calcolata, dallo Spider, la posizione media: per questo motivo il calcolo del voto finale è eseguito alla fine delle operazioni di minig. Il package ServerMiner contiene le seguenti classi:

- DocumentoPreProcessato:

  è la classe che gestisce il documento preprocessato e ne effettua l'analisi semantica, aggiornandone il relativo link. Implementa il SEMANTIC GRADER e il SYNTACTIC GRADER.

- MinerPages:

  è' la classe che implementa e gestisce il mining delle pagine. Ogni oggetto di tale classe, ovvero un nuovo Thread, assume la gestione di un elenco di documenti, preoccupandosi di effettuarne il mining. Il Thread esaurisce il suo compito quando sono stati elaborati tutti i documenti gestiti.

- QueryMiner:

  Gli oggetti di tale classe gestiscono una data query con l'ottica del mining. Dal DB del sistema vengono caricati esclusivamente i parametri necessari

per effettuare il mining, ovvero i parametri per i voti. Inoltre implementa le operazioni del GLOBAL GRADER.

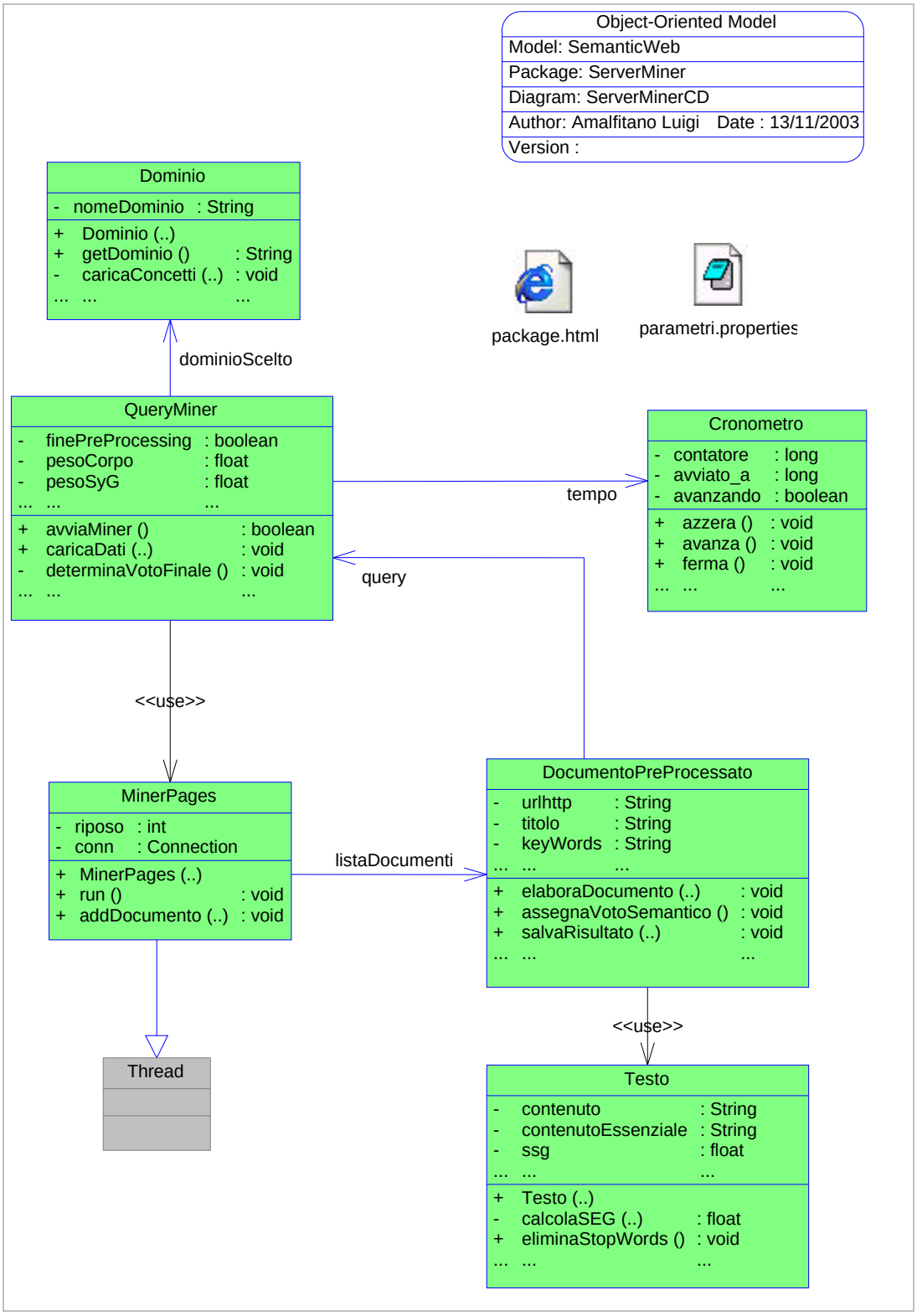

Figura 5.26: diagramma delle classi per il Server Miner

Dal diagramma delle classi si noti l'associazione della classe QueryMiner con la classe Cronometro (tempo): è importante per poter calcolare i tempi di elaborazione.

La classe QueryMiner risulta associata anche alla classe Dominio (dominioScelto): rappresenta il dominio selezionato per la query di cui si vuole fare il minig.

La classe MinerPages risulta associata con la classe DocumentoPreProcessato: questo perché ha il compito di gestire l'elaborazione di un set di documenti (listaDocumenti).

La classe DocumentoPreProcessato è associata alla classe QueryMiner: ogni documento appartiene ad una data query, in questo caso quella di cui si è richiesta l'elaborazione di mining (query). Inoltre la classe DocumentoPreProcessato fa uso di oggetti Testo per l'elaborazione semantica.

La classe QueryMiner utilizza un insieme variabile di oggetti MinerPages: uno per ogni nuovo accesso alla base dati che restituisce un elenco di documenti preprocessati da elaborare.

Il file package.html server per la documentazione e il file parametri.properties permette di impostare i parametri del server.

LA CLASSE DOCUMENTOPREPROCESSATO

La classe DocumentoPreProcessato svolge l'analisi semantica per un dato documento preprocessato. La fase di costruzione dell'oggetto prevede di assegnare, oltre al link, le parti fondamentali della pagina web, così come sono state determinate dal Server PreProcessor. Tali parti, lo ricordiamo, sono:

- il titolo;
- la descrizione;
- l'elenco delle parole chiavi;
- il corpo;

Dopo aver definito l'oggetto sarà possibile eseguire l'analisi semantica su ciascuna parte del documento, effettuare la media del SSG e del SeG e salvare i risultati nell'entità link. Tali operazioni vengono richieste con il messaggio elaboraDocumento() che invoca i metodi:

- assegnaVotoSemantico(), per assegnare i voti SSG e SeG;
- salvaRisultato(), per salvare i risultati nella base dati.

Il metodo assegnaVotoSemantico() utilizza oggetti Testo, ciascuno per ogni parte del documento. Gli oggetti Testo hanno la funzione di calcolare, per un dato testo, i voti semantici. Tali voti verranno prelevati, poi, mediante i metodi get (per maggiori dettagli si rimanda alla classe Testo). In ultimo, il metodo assegnaVotoSemantico effettua la media tra i quattro voti SeG e i quattro voti SSG; i pesi verranno prelevati dall'oggetto query associato.

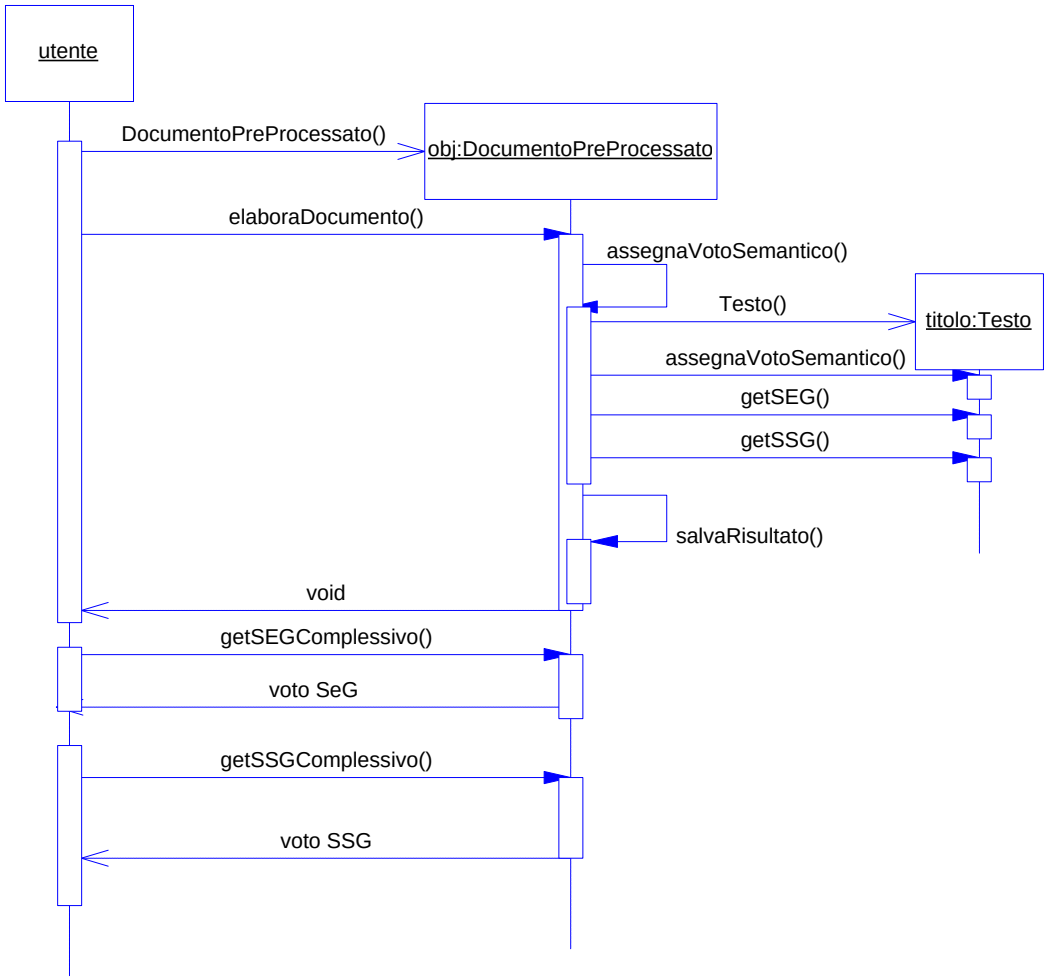

Figura 5.27: utilizzo di DocumentoPreProcessato per l'elaborazione semantica.

Si noti che nella figura 5.27 viene indicato solo l'oggetto testo relativo al titolo. Questo schema è esemplificativo, nella realtà ogni parte del documento avrà un proprio oggetto testo. Il metodo elabora documento non restituisce alcun valore: i risultati ven-

gono direttamente salvati nella base dati; tuttavia, per conoscere i voti assegnati al documento è possibile utilizzare il metodo getSEGComplessivo() e il metodo getSSGComplessivo().

LA CLASSE MINERPAGES

La classe MinerPages ha il compito di gestire l'elaborazione di un insieme di documenti. Ogni oggetto di tale classe rappresenta un nuovo Thread, appartenente ad un gruppo e creato da un oggetto della classe QueryMiner. All'atto della creazione dell'oggetto sarà necessario assegnare il nome del Thread e del gruppo di appartenenza. Inoltre, sempre all'atto della creazione viene definito un elenco vuoto che, in una fase successiva verrà riempito da oggetti DocumentoPreProcessato.

Per utilizzare l'oggetto della classe MinerPages sarà quindi necessario:

1. crearlo, definendo le caratteristiche del thread;
2. con il metodo addDocumento() assegnare un documento preprocessato all'oggetto MinerPages;
3. dopo aver assegnato tutti gli oggetti avviare il thread con il metodo start();

Tramite il messaggio addDocumento è possibile inviare all'oggetto MinerPages le parti di un documento preprocessato (come stringa) ed il link associato. Sarà poi il metodo addDocumento() che si preoccuperà di creare un nuovo oggetto della classe DocumentoPreProcessato e di inserirlo nell'elenco dei documenti. Il metodo addDocumento() dovrà essere utilizzato per ogni nuovo documento preprocessato elaborato dal preprocessor ed inserito nella base dati del sistema.

Dopo aver riempito l'elenco dei documenti è possibile avviare il thread MinerPages con il metodo start(): verrà avviato il mining sui documenti caricati. I documenti verranno elaborati in maniera sequenziale.

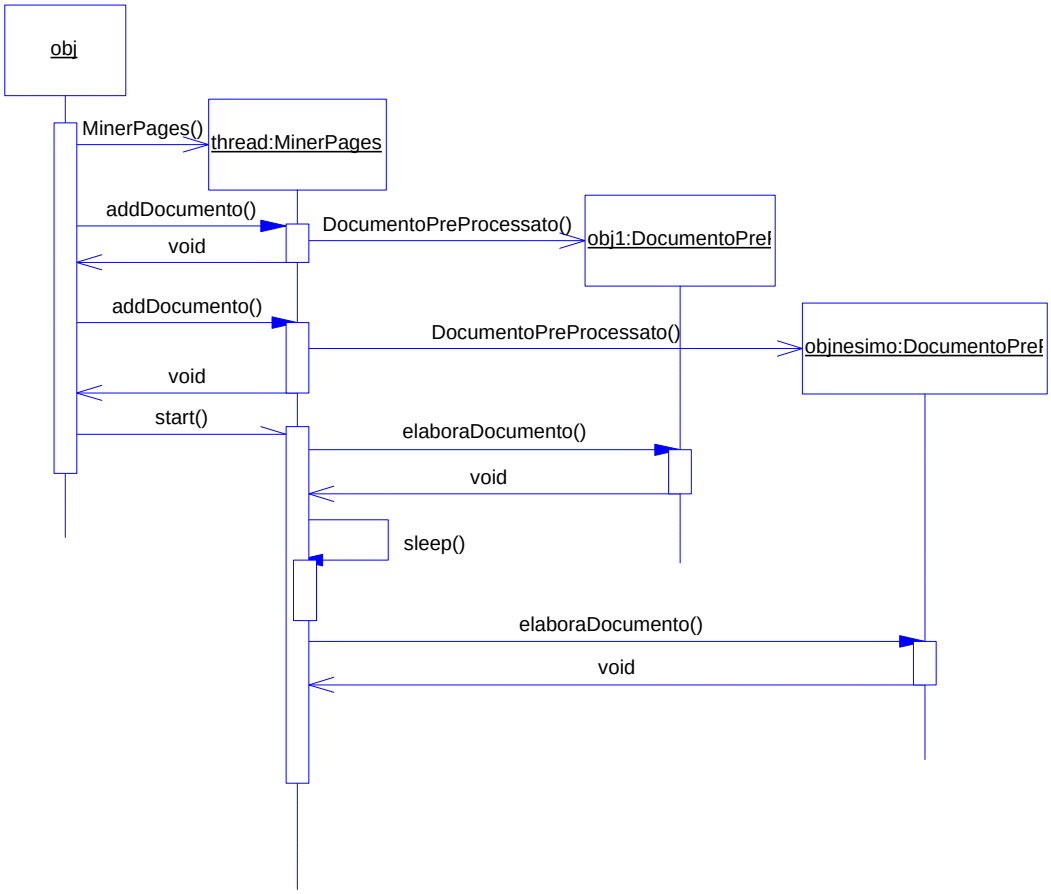

Figura 5.28: schema sequenziale di un oggetto della classe MinerPages.

Tra l'elaborazione di due documenti è possibile sospendere per alcuni millisecondi il thread corrente per dare spazio ad altri thread MinerPages

Il Thread terminerà il suo lavoro quando saranno stati elaborati tutti i documenti pre-processati. Alla fine, quindi, la base dati conterrà nella relazione Link i voti assegnati ai relativi documenti elaborati.

LA CLASSE QUERYMINER

La classe QueryMiner ha lo scopo di avviare il mining per una data interrogazione e di gestire l'accesso a nuovi documenti da elaborare.

Per eseguire tali compiti un oggetto della classe ha bisogno dell'identificatore della query, dei pesi associati ai singoli voti e alle varie parti del documento: è necessario accedere alla base dati per reperire tali informazioni. Inoltre l'oggetto QueryMiner necessita di una istanza della classe Dominio. L'istanza è relativa al dominio scelto per la data query. Con il costruttore dell'oggetto è possibile associare l'identificatore della query e il dominio. Successivamente, con il metodo caricaParametriQuery() è possibile reperire le altre informazioni dalla base dati. Naturalmente, è richiesto che l'identificatore si riferisca realmente ad una query presente nella base dati del sistema.

Le operazioni di mining vengono attivate mediante il messaggio avviaMiner(), che può essere invocato appena dopo aver definito l'oggetto e invocato il metodo caricaParametriQuery(). Tale messaggio è non bloccante: crea immediatamente un nuovo Thread che effettua tutte le operazioni relative al mining dei documenti preprocessati.

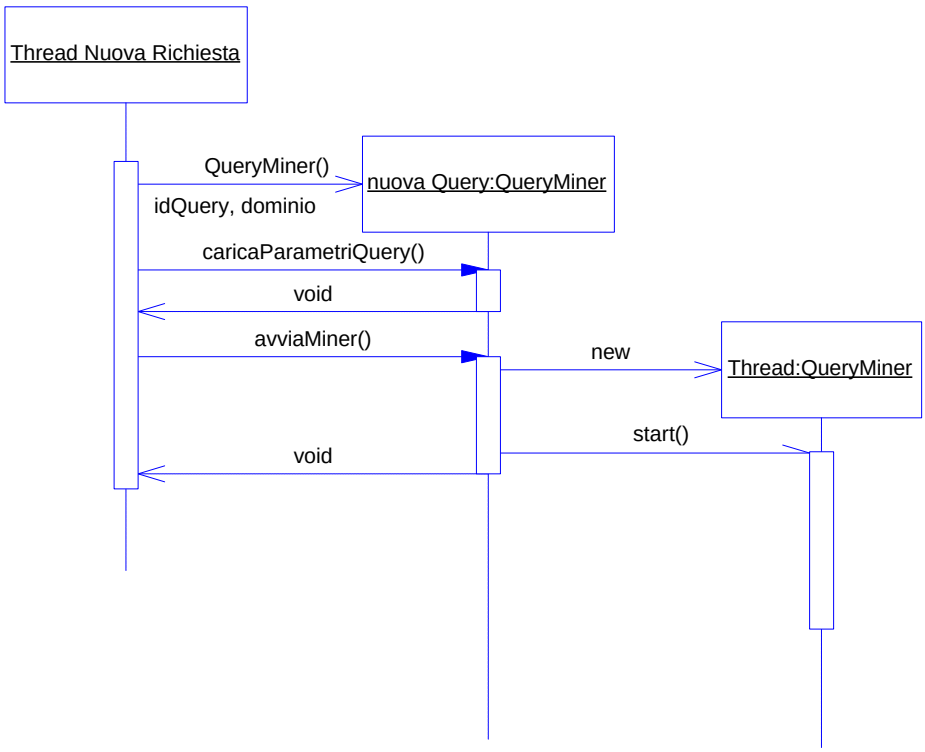

Figura 5.29: avvio non bloccante delle operazioni di minig.

In questo modo quando l'interfaccia Server riceve una chiamata dal client, crea si un nuovo Thread (Thread Nuova Richiesta), come prevede il modello client-server, ma questo thread utilizza poi il metodo avviaMiner() su un oggetto QueryMiner, avviandone l'elaborazione e rilasciando immediatamente il client.

Il cuore della gestione del mining risiede nel Thread creato mediante il messaggio avviaMiner: il Thread deve caricare dalla base dati nuovi documenti preprocessati fino a quando:

1. il server preprocessor non termina la propria elaborazione (per la query data);
2. non ci sono più documenti preprocessati da prelevare;

Tutto ciò viene realizzato all'interno di un ciclo che verrà eseguito fino a quando lo stato di elaborazione del preprocessor è falso, ovvero fino a quando è in attività. Il ciclo comprende le seguenti operazioni:

a. viene settato lo stato del Preprocessor, mediante il metodo setFinePreProcessing();
b. vengono prelevati eventuali nuovi documenti preprocessati dalla base dati, mediante il metodo caricaDati();
c. si controlla lo stato di elaborazione del Preprocessor, con il metodo getFinePreProcessing().

L'operazione principale del ciclo è rappresentata dal metodo caricaDati(). Tale metodo ha accesso alla base dati ed effettua nell'ordine indicato le seguenti operazioni:

- blocco write per la tabella DocumentiPreProcessati;
- preleva tutti i nuovi documenti preprocessati elaborati dal PreProcessor (ossia quelli che per l'attributo **statoMiner** assumono valore falso);
- aggiorna lo stato dei documenti prelevati, assegnando per l'attributo statoMiner valore vero;
- sblocca la tabella DocumentiPreProcessati;
- se ci sono nuovi documenti:

- crea un thread MinerPages;
- per ogni nuovo documento invoca il messaggio addDocumento();
- avvia il thread MinerPages;

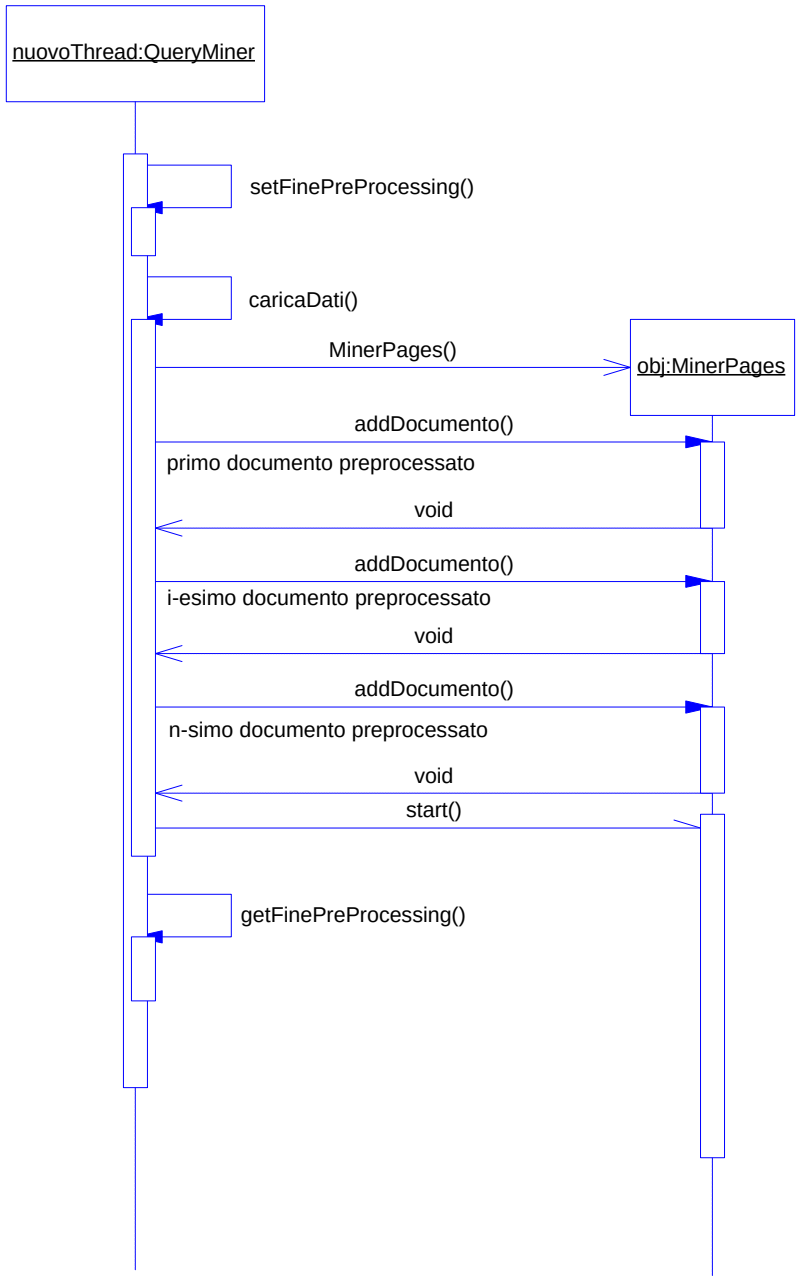

Figura 5.30: avvio del mining per un elenco di nuovi documenti preprocessati.

Il metodo setFinePreProcessing() ha accesso alla base dati del sistema per controllare, nell'entità stato, l'attività del server preprocessor per la query di cui si sta effettuando

il mining: se l'attività è terminata l'indicatore di stato associato assumerà valore vero, altrimenti assumerà valore falso.

Il metodo getFinePreProcessing() ha il compito di leggere il valore assegnato all'indicatore di stato relativo al preprocessing.

E' importante notare l'ordine delle operazioni all'interno del ciclo: verifica le condizioni 1 e 2 di esecuzione del Thread.

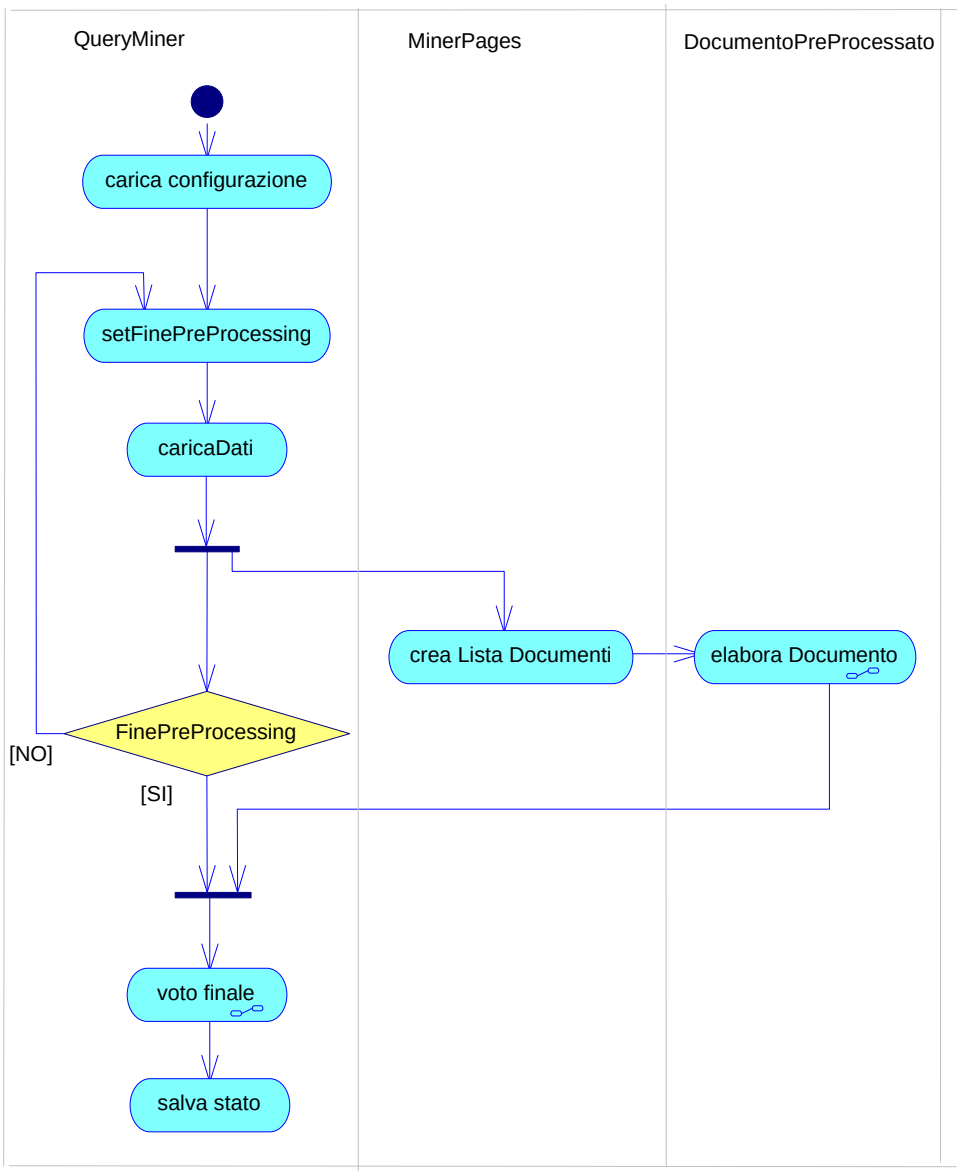

Figura 5.31: attività di gestione delle operazioni di mining.

Quando il preprocessor ha terminato la propria elaborazione e sono stati prelevati tutti i documenti preprocessati dalla base dati, è necessario attendere che i vari thread creati dal metodo caricaDati(), in chiamate successive, abbiano cessato la propria attivi-

tà; ciò indica il fatto che sono stati analizzati semanticamente tutti i documenti presenti nella base dati.

Mediante il metodo determinaVotoFinale() verranno determinati i voti finali di tutti i link (Global Grader). Tale operazione viene fatta direttamente nella base dati creando un unico comando SQL (update).

L'attività di mining termina, quindi, invocando il messaggio salvaStatoElaborazione(). Tale metodo consente di salvare nella tabella stato della base di dati l'informazione di fine attività del server Miner (SM) per la query data e le informazioni di log relative ai tempi di elaborazione.

GESTIONE DEI THREAD MINERPAGES E TEMPIFICAZIONE

Da quanto detto nei paragrafi precedenti, si può constatare che l'attività di mining, come quella di preprocessing, dei documenti appartenenti ad una data query è condotta da un insieme di thread che vengono creati ogni volta che si accede alla base dati e si individuano nuovi documenti preprocessati.

Per un funzionamento efficiente del sistema sarebbe necessario che i documenti prelevati dalla base dati per l'elaborazione siano distribuiti quanto più uniformemente possibile tra i vari thread che possono essere creati. Tuttavia, la distribuzione dei documenti fra i vari thread non è molto semplice poiché essi vengono inseriti nel sistema in tempi successivi, che dipendono molto dall'attività di spidering e di preprocessing.

Si è pensato, quindi, di stabilire una tempificazione sul metodo caricaDati(): esso verrà invocato con una certa frequenza che è possibile stabilire e impostare in maniera dinamica. Si definisce come tempo di caricamento fra i dati (tcd) l'intervallo fra due chiamate al metodo caricaDati(). Tale valore è espresso in millisecondi e viene impostato in un file di proprietà.

Dopo aver avviato tutti i thread, quello principale, di tipo QueryMiner, si pone in attesa della loro terminazione: con una certa frequenza è possibile controllarne il loro stato di terminazione. L'intervallo, in millisecondi, fra due controlli successivi sullo stato dei thread viene definito come tempo fra thread (tft) e può essere impostato in maniera dinamica.

I due valori, tcd e tfd, vengono definiti nel file di proprietà (parametri.properties) presente nel package e vengono caricati ogni volta che si invia il messaggio caricaDati(), come indicato in figura 5.31. Anche il Miner, come il PreProcessor può essere oggetto di controllo a retroazione per garantire un'efficienza minima.

Capitolo 6

Interfacce

6.1 Architettura RMI del sistema
6.2 L'interfaccia utente

Per poter utilizzare le funzionalità degli oggetti appartenenti al sistema, il client dovrà rivolgere specifiche richieste a tre applicazioni che gestiranno gli oggetti dei relativi package (spider, preprocessor, miner). Le richieste dovranno soddisfare il modello di comunicazione prefissato e dovranno nascondere all'utente tutte le caratteristiche relative al funzionamento dei vari oggetti del sistema. In pratica, il client richiederà l'utilizzo di oggetti remoti ai quali invierà messaggi specifici, per l'attivazione delle varie operazioni.

L'utente interagisce con il sistema tramite un normale browser: sarà necessario che la query inserita venga memorizzata e successivamente venga passato il controllo al client del sistema, che risiederà, di solito, sul web server.

6.1 ARCHITETTURA RMI DEL SISTEMA

Il modello RMI richiede che le informazioni necessarie dell'oggetto remoto debbano essere separate nelle interfacce lato client e nelle implementazioni lato server.

Il programma client deve poter manipolare gli oggetti server, ma non dispone di alcuna copia di tali oggetti. Gli oggetti si trovano sul server. Il codice client deve anche sapere cosa può fare con tali oggetti. Le loro capacità sono espresse in un'interfaccia condivisa tra client e server e che, quindi, si trova contemporaneamente su entrambe le macchine.

Riconsideriamo quindi il modello di comunicazione tra client e server esposto nel capitolo 3.

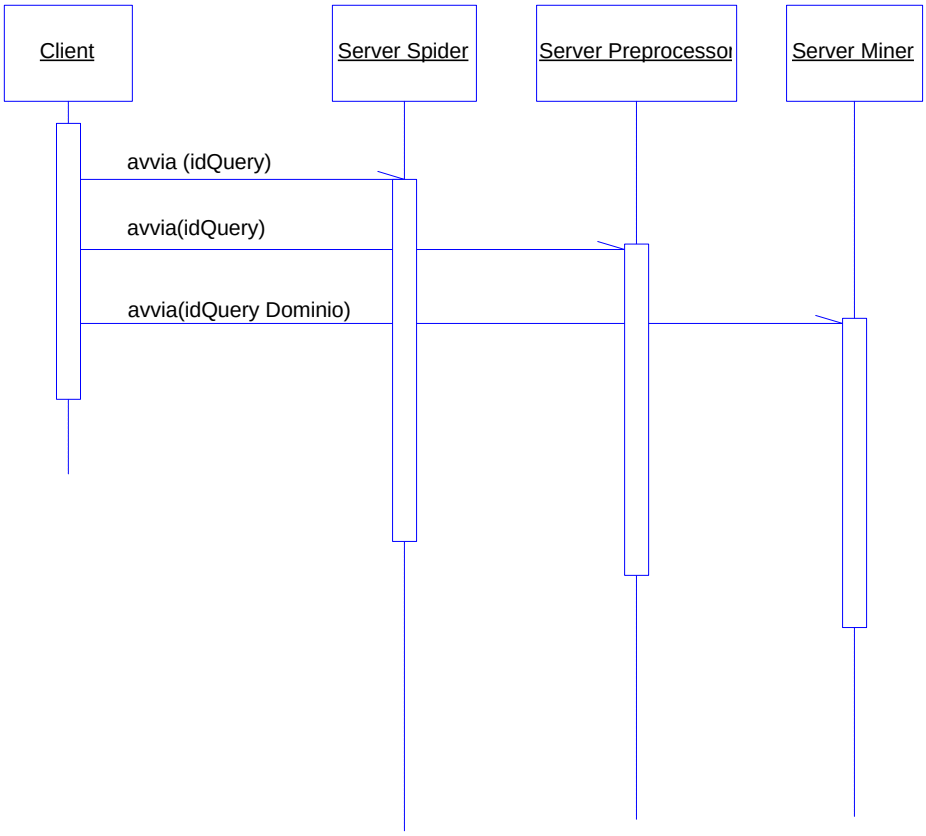

Figura 6.1: modello di comunicazione del motore di ricerca semantico.

Si osservi che, per ogni server, l'interfaccia di comunicazione è rappresentata dal metodo avvia(): tale messaggio dovrà essere dichiarato nell'interfaccia condivisa.

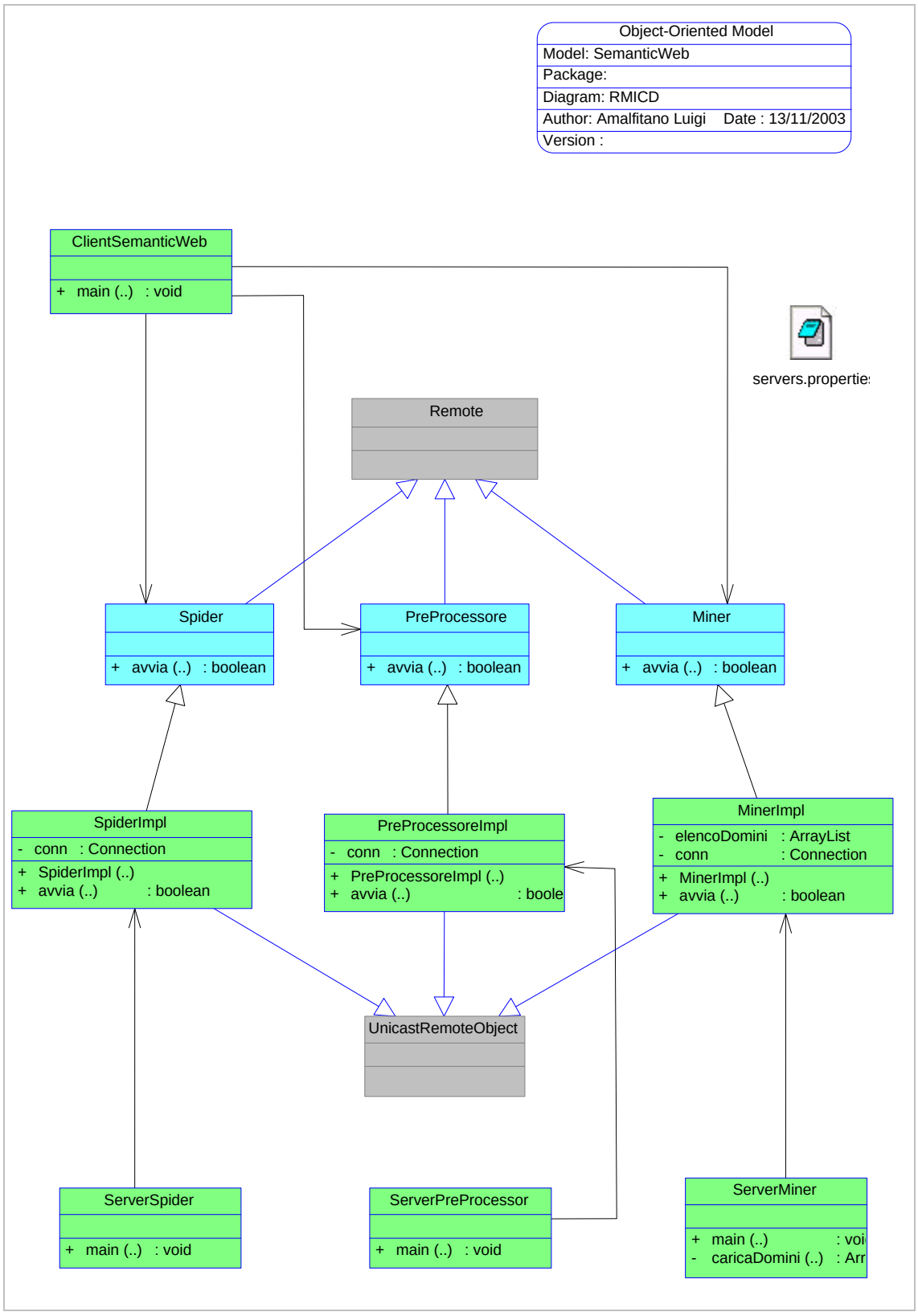

Figura 6.2: diagramma delle classi relativo agli oggetti remoti dei server del sistema.

Per il sistema motore di ricerca semantico è possibile definire le seguenti interfacce:

- Spider:

  rappresenta l'oggetto remoto reso disponibile dal programma server che gestisce lo spidering. Questa interfaccia prevede un unico metodo, avvia(idQuery), che consente di avviare per una data query, precedentemente memorizzata, l'operazione di spidering.

- PreProcessore:

  rappresenta l'oggetto remoto reso disponibile dal programma server che gestisce il preprocessing dei documenti. Anche questa interfaccia prevede il metodo avvia(idQuery) che consente di avviare il preprocessing per una data query presente nella base dati.

- Miner:

  rappresenta l'oggetto remoto reso disponibile dal programma server che gestisce il mining dei documenti preprocessati. L'interfaccia prevede il metodo avvia(idQuery, nomeDominio) che consente di avviare il mining per una data query presente nella base dati e riferita ad un dato dominio.

Ciascuna interfaccia dovrà essere implementata da un'opportuna classe:

- SpiderImpl:

  implementa l'interfaccia Spider. Il metodo avvia() fa riferimento al package ServerSpider: crea l'oggetto q, appartenente alla classe Query, lo inizializza e trasmette il messaggio q.avvia(), terminando.

- PreProcessoreImpl:

  implementa l'interfaccia PreProcessore. Il metodo avvia() fa riferimento al package ServerPreprocessor: crea l'oggetto query, appartenente alla classe QueryPrePro e trasmette il messaggio avviaPreProcessing(), terminando.

- MinerImpl:

  implementa l'interfaccia Miner. Il metodo avvia() fa riferimento al package ServerMiner: ricerca l'istanza all'oggetto Dominio relativa al nome domi-

nio, crea l'oggetto query, appartenente alla classe QueryMiner, lo inizializza e trasmette il messaggio avviaMiner(), terminando.

E' necessario precisare che, per accedere ad un oggetto remoto che si trova sul server, il client ha bisogno di un oggetto stub locale. Il metodo più comune per richiederlo consiste nel chiamare un metodo remoto di un altro oggetto server e ottenere come valore di ritorno un oggetto stub. Per localizzare il primo oggetto, la libreria RMI di Sun fornisce un servizio di registro all'avvio (rmiregistry). Il programma server registra gli oggetti con il servizio di registro all'avvio (viene fornito un nome simbolico) e il client recupera gli stub per quegli oggetti. Tale operazione dovrà essere eseguita da ogni programma server.

I programmi server, che implementeranno il sistema, elencati di seguito, potranno essere esegui su macchine diverse. Ciascuno di esso eseguirà una serie di attività e alla fine si porrà in attesa: per ogni richiesta inviata al programma server verrà creato un thread che eseguirà le operazioni appartenenti al relativo messaggio.

- L'applicazione ServerSpider:
  1. crea una connessione alla base dati, necessaria per tutte le elaborazioni di spidering;
  2. crea un oggetto ServerImpl, passandogli la connessione creata;
  3. lo registra con il nome di Spider;
  4. si pone in attesa delle richiesta da parte del client.

- L'applicazione ServerPreProcessor:
  1. crea una connessione alla base dati, necessaria per tutte le elaborazioni di preprocessing;
  2. crea un oggetto PreProcessoreImpl, passandogli la connessione creata;
  3. lo registra con il nome di PreProcessore;
  4. si pone in attesa delle richiesta da parte del client.

- L'applicazione ServerMiner:
  1. crea una connessione alla base dati, necessaria per tutte le elaborazioni di minig;
  2. definisce l'elenco degli oggetti Dominio;

3. crea un oggetto PreProcessoreImpl, passandogli la connessione creata e l'elenco degli oggetti dominio;
4. registra l'oggetto con il nome di Miner;
5. si pone in attesa delle richiesta da parte del client.

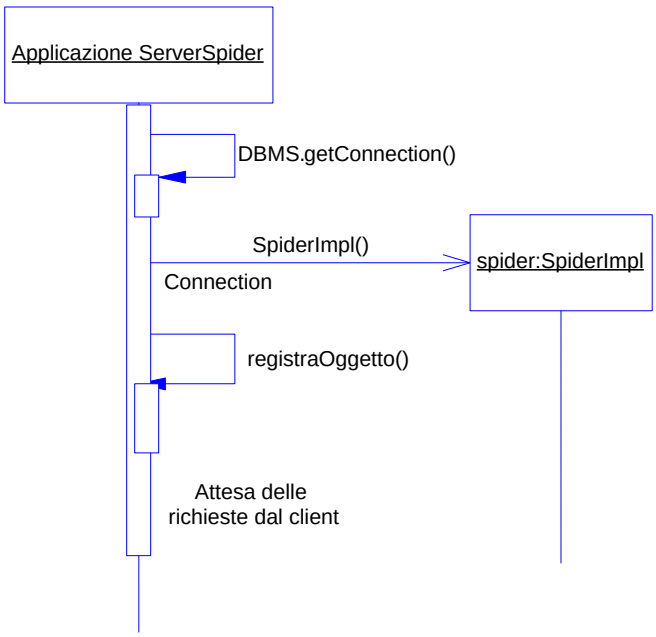

Figura 6.3: attivazione dell'applicazione Server relativa allo Spider.

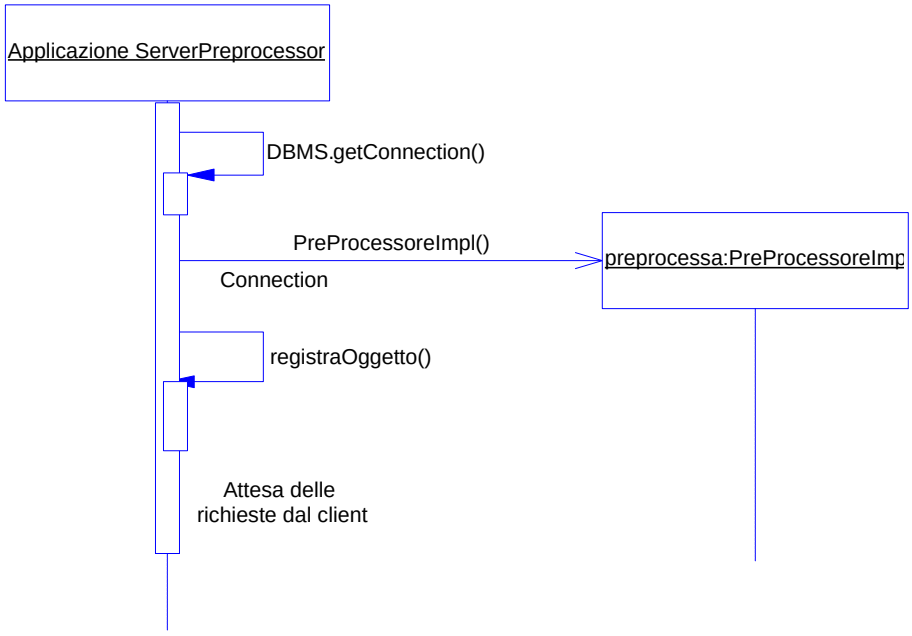

Figura 6.4: attivazione dell'applicazione Server relativa al Preprocessor.

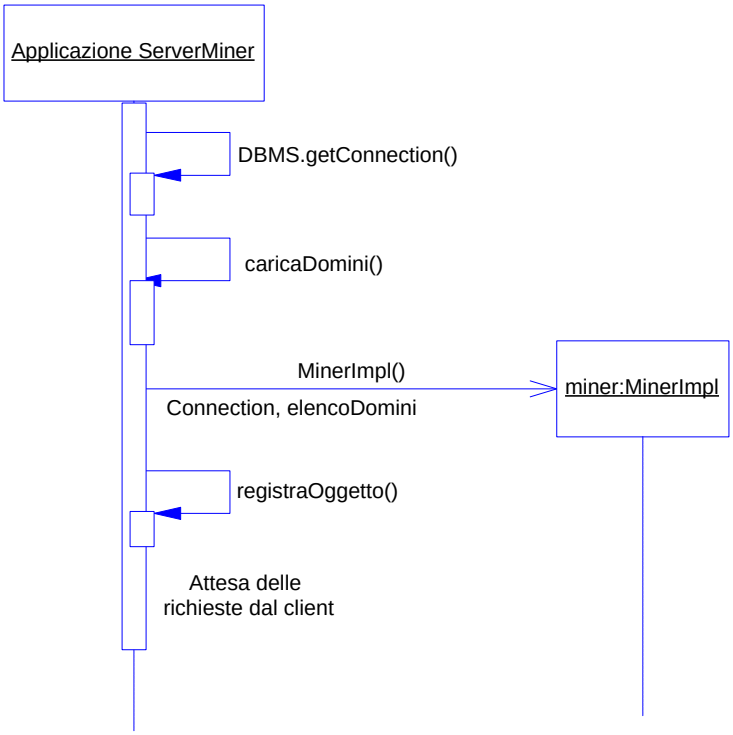

Figura 6.5: attivazione dell'appicazione server relativa al Miner.

Il compito del programma client sarà quello di ottenere uno stub per accedere all'oggetto server: è necessario indicare l'indirizzo rmi e il nome dell'oggetto. Successivamente sarà possibile invocare i relativi metodi. Quindi, il client dovrà:

1. definire un oggetto, serverSpider, relativo all'interfaccia Spider;
2. caricare lo stub Spider da rmi://server:1099/Spider;
3. definire un oggetto, serverPreProcessor, relativo all'interfaccia PreProcessore;
4. caricare lo stub PreProcessore da rmi://server:1099/PreProcessore;
5. definire un oggetto, serverMiner, relativo all'interfaccia Miner;
6. caricare lo stub Miner da rmi://server:1099/Miner;

Per evitare di dover modificare il codice del client quando si intende spostare i server, le informazioni relative agli indirizzi sono state inserite in un file di proprietà, servers.properties.

Nella figura 6.6 si mostra il modo in cui il client carica dinamicamente i vari stub dai relativi server.

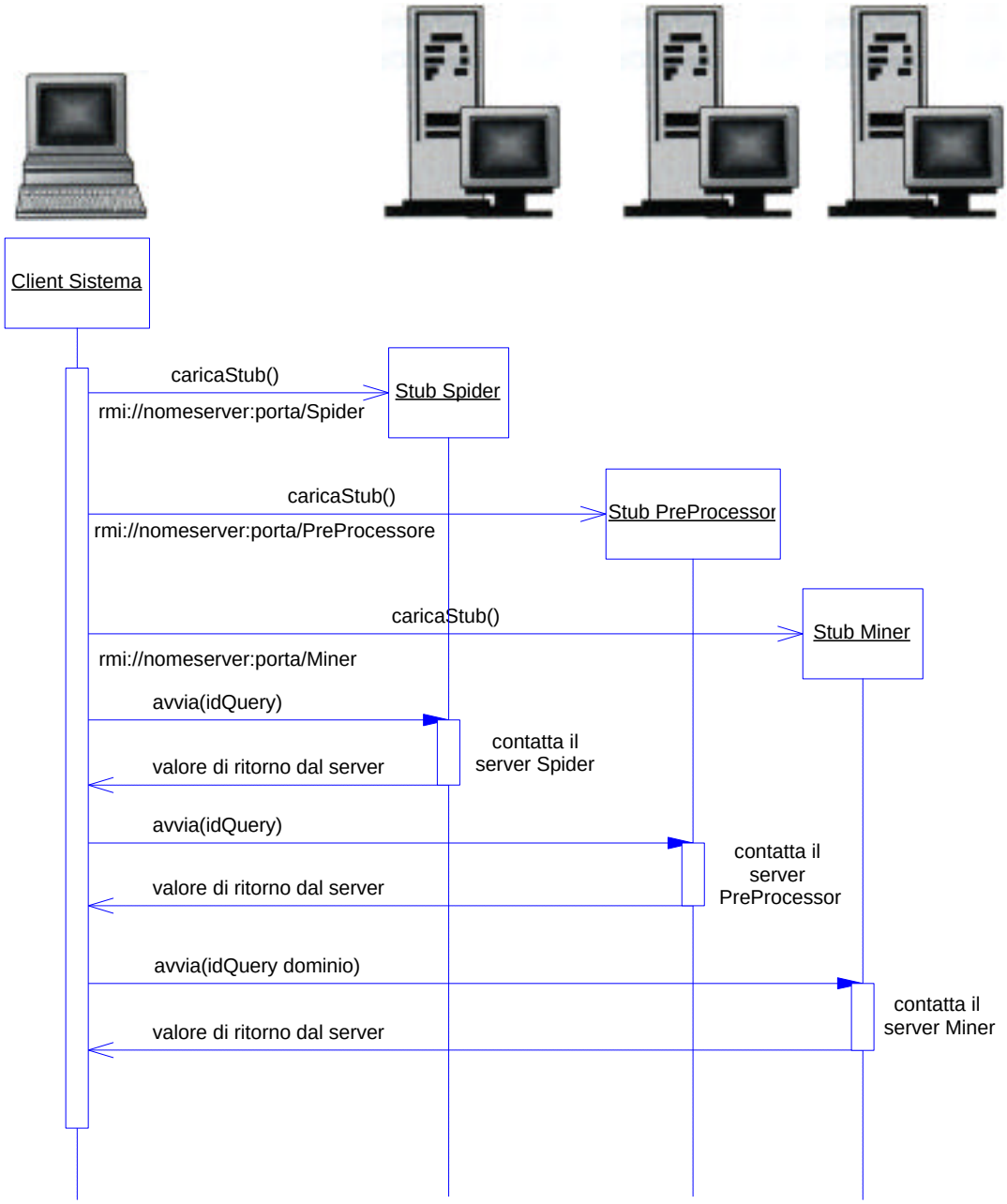

Figura 6.6: caricamento dinamico degli stub dai relativi server ed invio dei relativi metodi.

Dopo aver caricato gli stub sarà possibile, per ogni query registrata di cui si vorrà eseguire la ricerca semantica, eseguire i seguenti metodi non bloccanti:

1. serverSpider.avvia(idQuery);
2. serverPreProcessor.avvia(idQuery);
3. serverMiner.avvia(idQuery,nomeDominio);

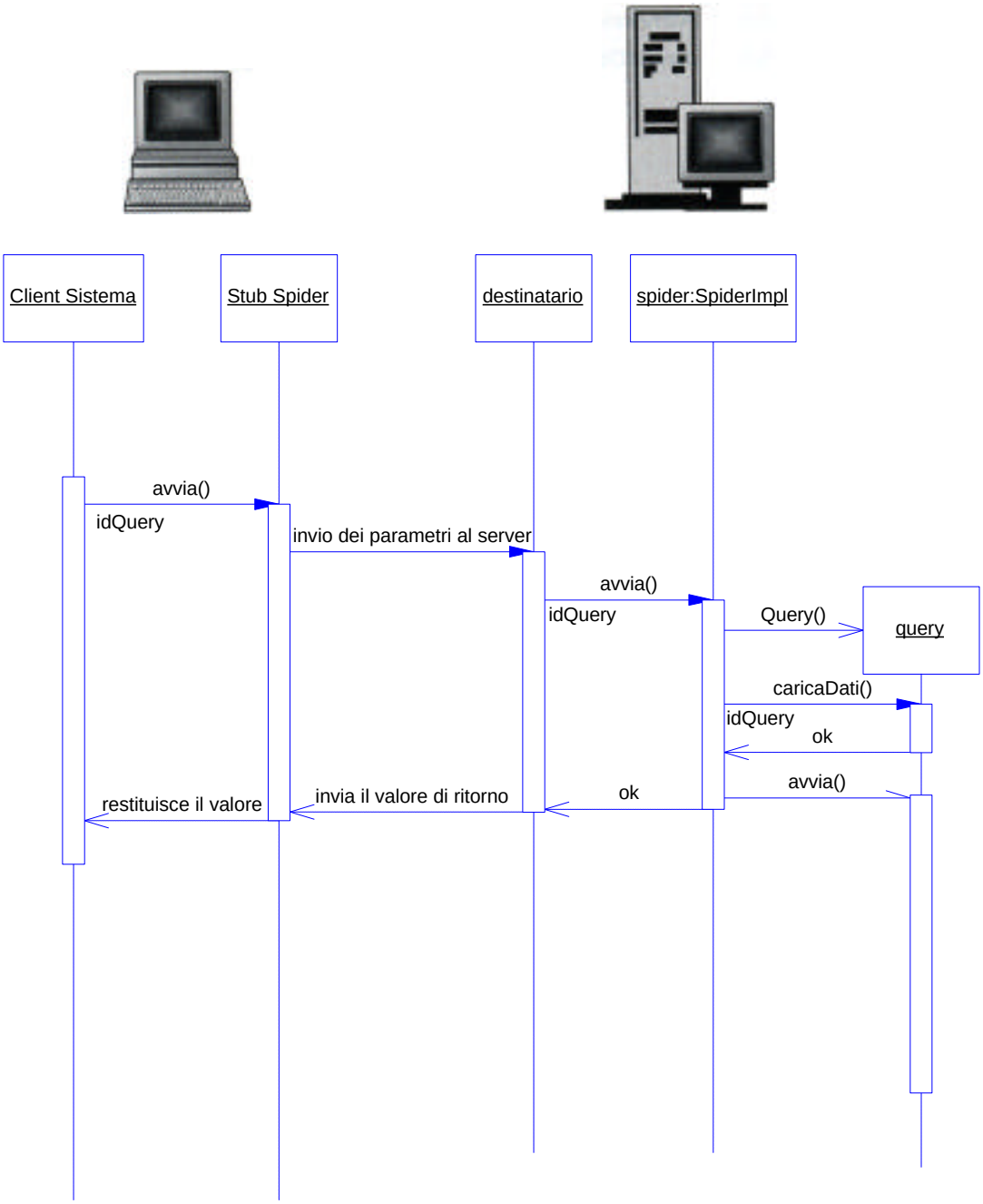

Figura 6.7: trasmissione del metodo avvia() all'oggetto remoto relativo allo spider.

Nelle figura 6.7 si evidenzia la trasmissione del messaggo avvia() all'oggetto remoto relativo allo spider. E' importante osservare che la classe di implementazione della interfaccia spider realizza nel metodo di avvia tutto ciò che è stato indicato per la classe Query del package ServerSpider; in particolare si noti la chiamata avvia() che genera un nuovo thread (per maggiori dettagli si rimanda al capitolo 5).

Il funzionamento del messaggio relativo allo stub del preprocessing e del miner è analogo e, anche per essi, il metodo avvia() creerà, rispettivamente, un oggetto Quer-

yPrePro e un oggetto QueryMiner, e vi invierà poi un messaggio non bloccante che genererà un nuovo thread.

6.2 INTERFACCIA UTENTE

L'interfaccia utente del sistema è rappresentata da un insieme di pagine web, attraverso le quali è possibile:

- ✓ sottoporre nuove query al sistema;
- ✓ visualizzare i risultati per le query già elaborate;
- ✓ amministrare il sistema.

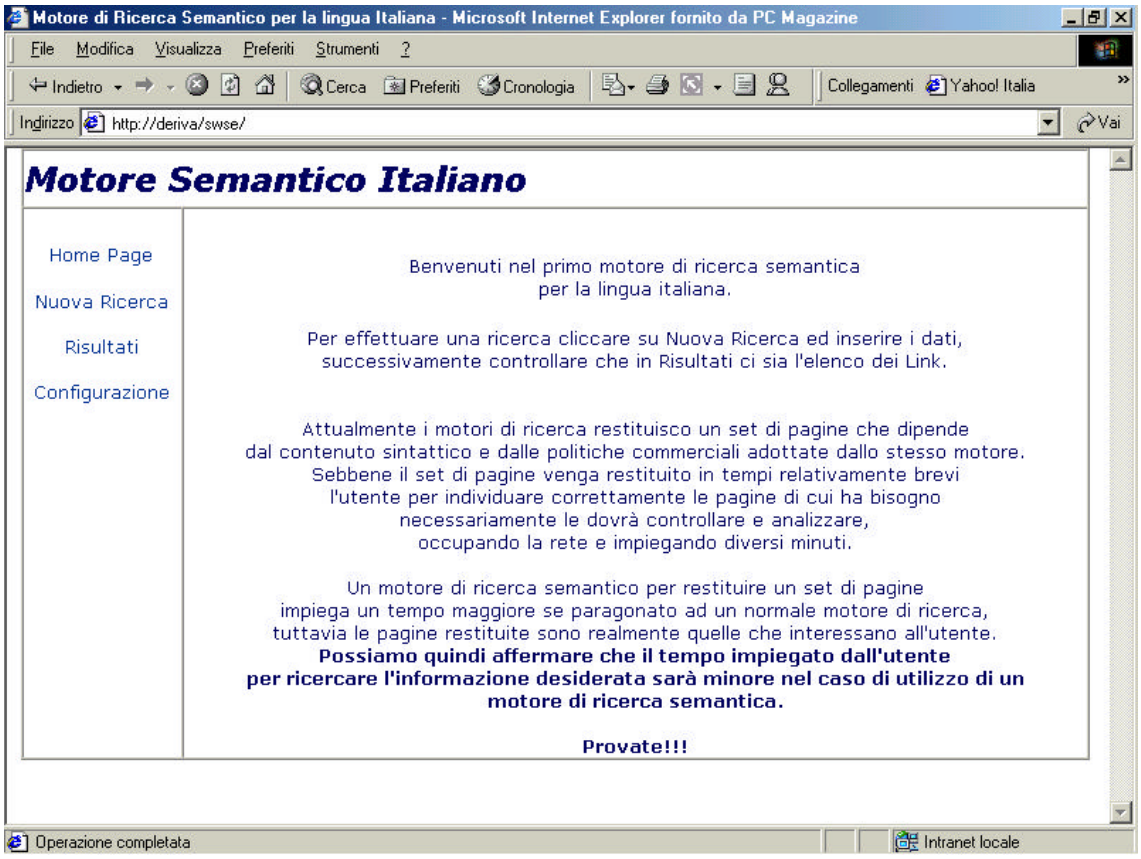

Figura 6.8: Home page e barra di navigazione del motore di ricerca.

NUOVA QUERY

Quando l'utente seleziona nuova ricerca dalla barra di navigazione uno script lato server, in codice php, provvede a caricare direttamente dalla base dati i domini disponibili nell'ontologia e i motori di ricerca che è possibile utilizzare e sottoporrà all'utente un modulo da compilare. L'utente dovrà obbligatoriamente:

- ✓ inserire i vocaboli da ricercare (è indifferente se in minuscolo o in maiuscolo);
- ✓ selezionare il dominio;
- ✓ selezionare i motori di ricerca da utilizzare per lo spider.

Per i motori di ricerca uno script lato client calcolerà direttamente i pesi relativi, utilizzando l'indicazione sul peso assoluto inserito dall'amministratore; comunque, l'utente potrà sempre modificarli.

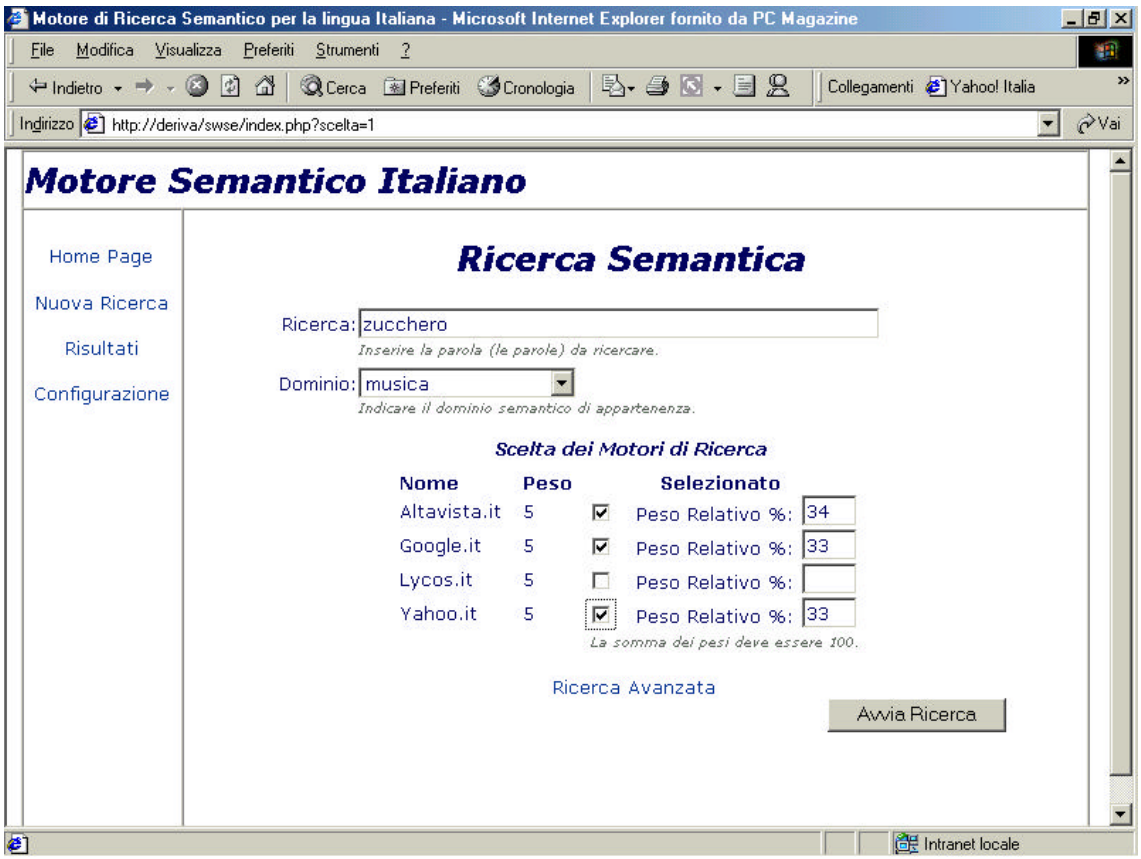

Figura 6.9a: ricerca semplice.

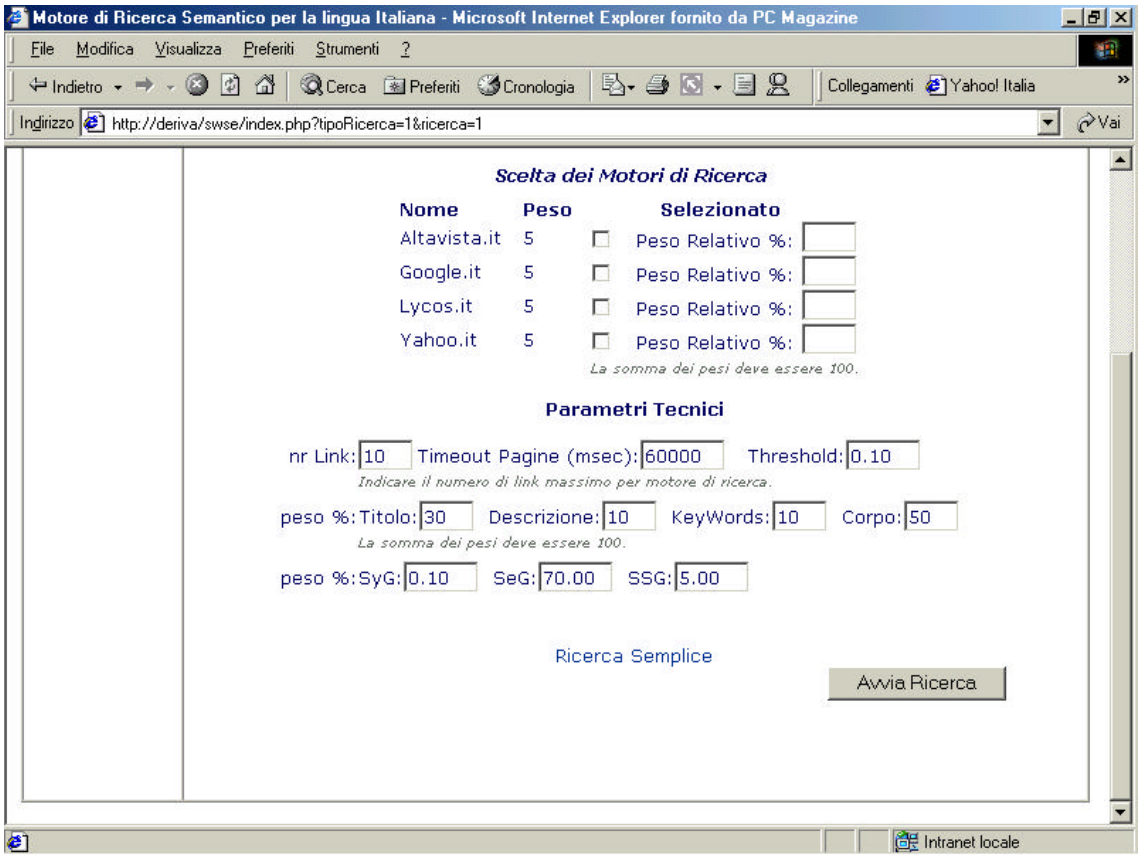

Figura 6.9b: ricerca avanzata.

L'utente potrà modificare, nella ricerca avanzata, anche alcuni parametri tecnici, che altrimenti sarebbero impostati per default:

- ✓ numero massimo di link per motori di ricerca;
- ✓ timeout in millisecondi per il download delle pagine;
- ✓ threshold;
- ✓ peso del titolo;
- ✓ peso della descrizione;
- ✓ peso delle parole chiavi;
- ✓ peso del corpo;
- ✓ peso del grado sintattico, SyG;
- ✓ peso del grado semantico, SeG;
- ✓ peso del grado sintattico-semantico,SSG.

All'atto dell'invio del modulo, tutti i dati inseriti dall'utente saranno controllati da uno script lato client e, solo se non ci saranno errori, verranno trasmessi al server (metodo POST).

L'invio di una nuova richiesta determinerà, in ordine, le azioni di seguito elencate.

A. Attivazione di uno script php che provvederà a salvare la query nella base dati del sistema, ricavandone l'indentificatore primario (idQuery).

B. Invio, da parte dello script php, dell'indificatore primario (idQuery) e del nome rappresentante il dominio scelto ad un'application server, ovvero una servlet che **si comporta da client del sistema**.

C. La servlet invierà i messaggi di avvia() ai relativi oggetti remoti; naturalmente si presuppone che i server del sistema siano attivi.

D. L'utente verrà informato che la trasmissione della ricerca è avvenuta con successo.

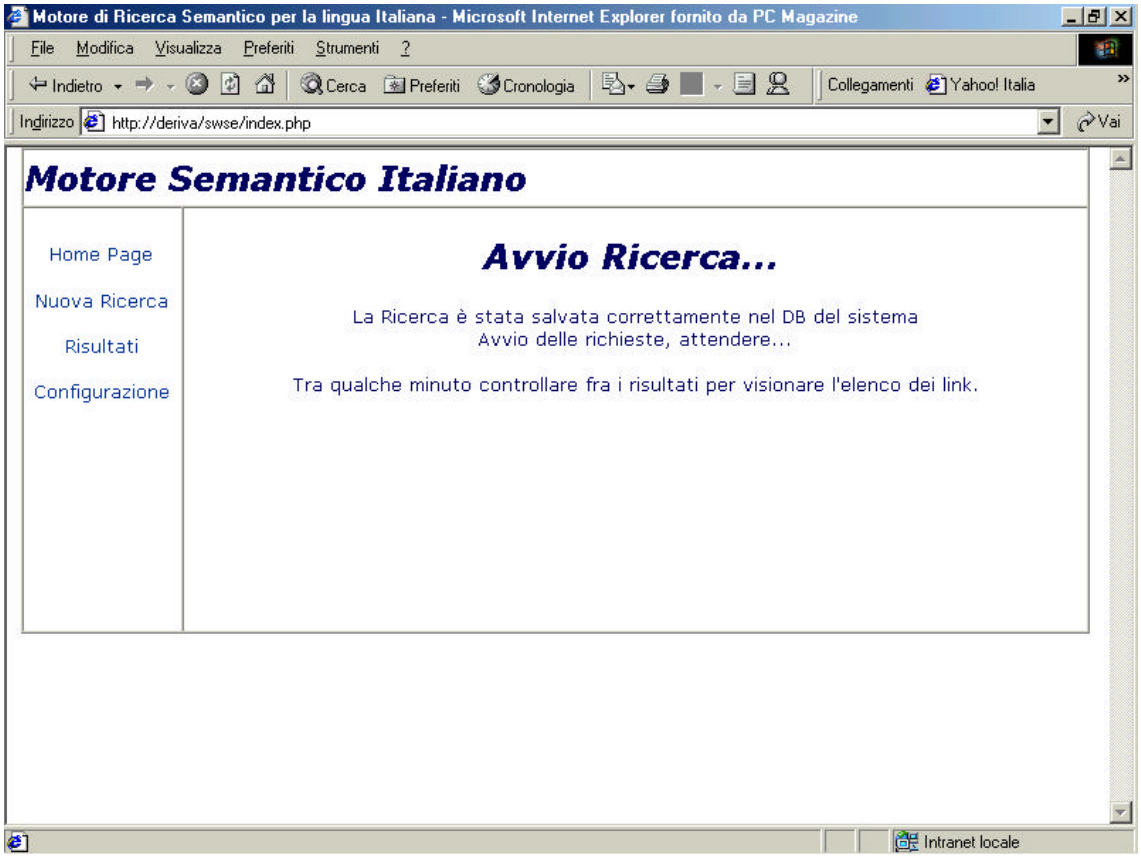

Figura 6.9c: inoltro e attivazione di una nuova ricerca al sistema.

L'utente, dopo aver inoltrato la query potrà prendere visione dei risultati: è importante notare che l'utente viene subito liberato dalla connessione al sistema.

Il tempo necessario per effettuare la ricerca semantica dipende dalla complessità della query, dal numero dei motori selezionati, dal numero massimo di link e dal supporto hardware di cui il sistema può beneficiare. Per questo motivo l'utente ha la possibilità di depositare la richiesta e di esaminarla più tardi.

VISUALIZZAZIONE DEI RISULTATI

Dalla barra di navigazione, selezionando risultati, sarà possibile esaminare i link individuati per ciascuna query sottoposta al sistema.

Il relativo script php accederà alla base dati, prelevando esclusivamente le query già elaborate e costruendo un elenco, che sarà trasmesso al browser.

Figura 6.10a: selezione di una query elaborata

L'utente potrà selezionare la query desiderata e attivare, quindi, un opportuno script php che preleverà dalla base dati tutti i link individuati dal motore di ricerca semantico.

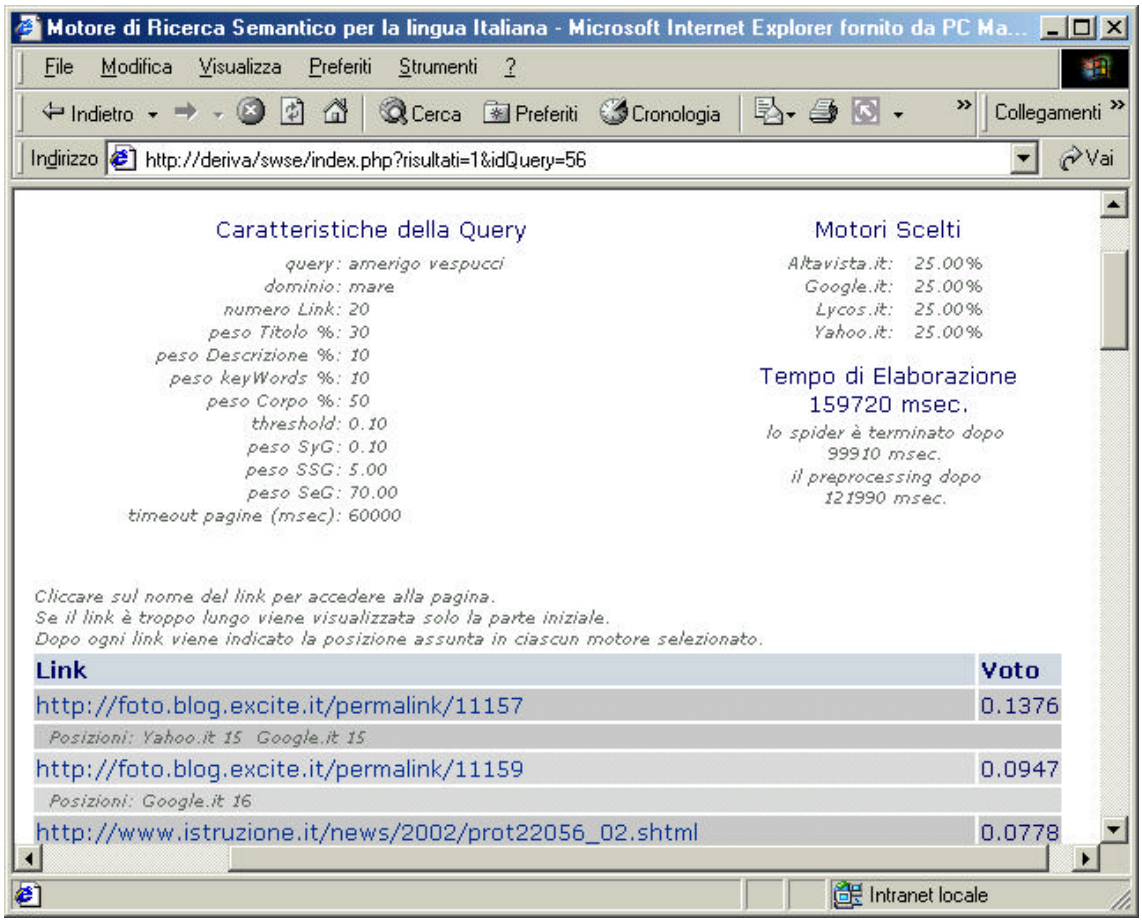

Figura 6.10b: visualizzazione dei risultati per la query zucchero.

I link verranno visualizzati in ordine di voto, in modo che l'utente si possa rendere conto dell'efficacia del nostro sistema.

MODIFICA E INSERIMENTO DI NUOVI MOTORI DI RICERCA

L'amministratore potrebbe avere l'esigenza di modificare le caratteristiche dei motori di ricerca inseriti nel sistema oppure di aggiungerne di nuovi.

Tali operazioni sono state rese disponibili creando un unico script php che in maniera estremamente semplice permette sia di effettuare modifiche che di aggiungere motori di ricerca. Infatti, l'amministratore, selezionando configurazione, accederà ad un'area protetta in cui gli verrà presentata una pagina web composta da un modulo e da un elenco dei motori presenti nel sistema.

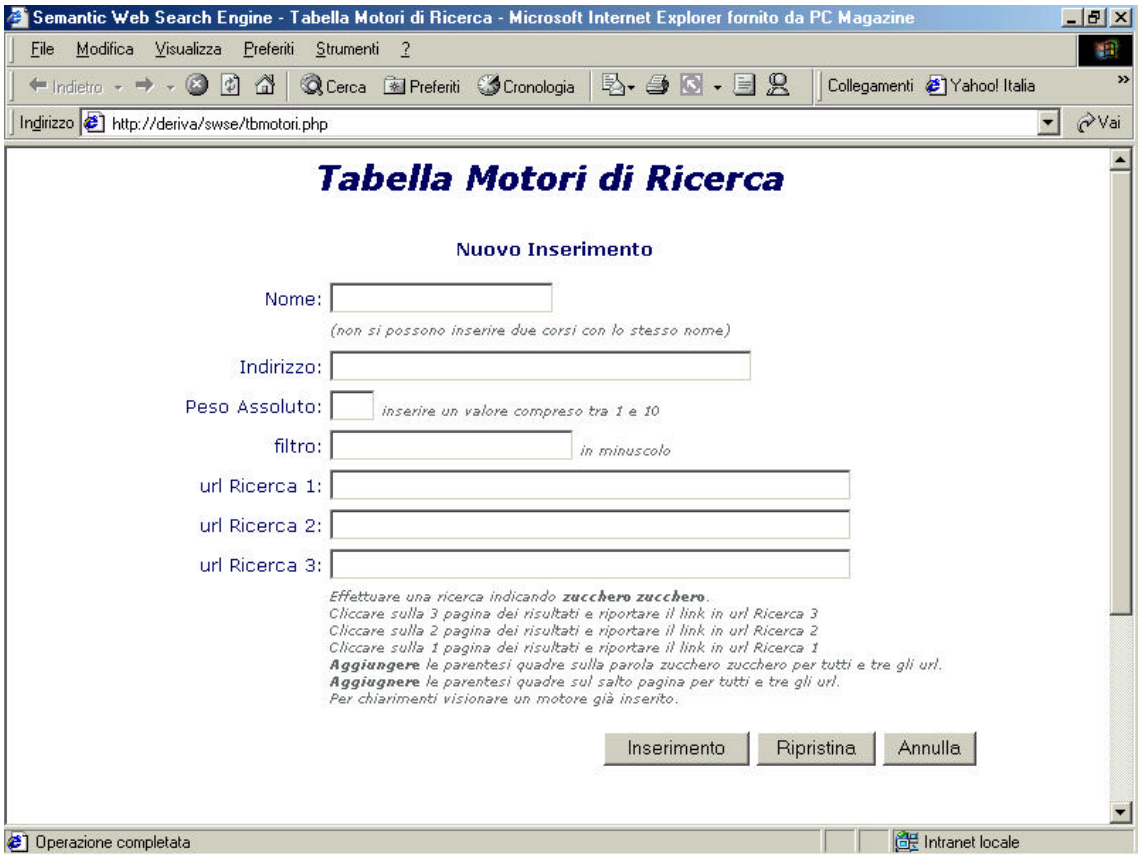

Figura 6.11a: inserimento di un nuovo motore di ricerca.

Per effettuare un nuovo inserimento basta compilare il modulo indicando:

- ✓ nome, simbolico, del motore di ricerca;
- ✓ indirizzo http associato;
- ✓ peso assoluto;
- ✓ filtro;
- ✓ url Ricerca 1, url Ricerca 1, url Ricerca 1;

Per gli url di ricerca si ricorda che devono essere conformi allo schema esposto nel capitolo 3.

I dati relativi al nuovo motore di ricerca verranno prima controllati da un client script, e, solo se corretti, verranno trasmessi ad uno script lato server, script php, che si preoccuperà di salvarli direttamente nella base dati.

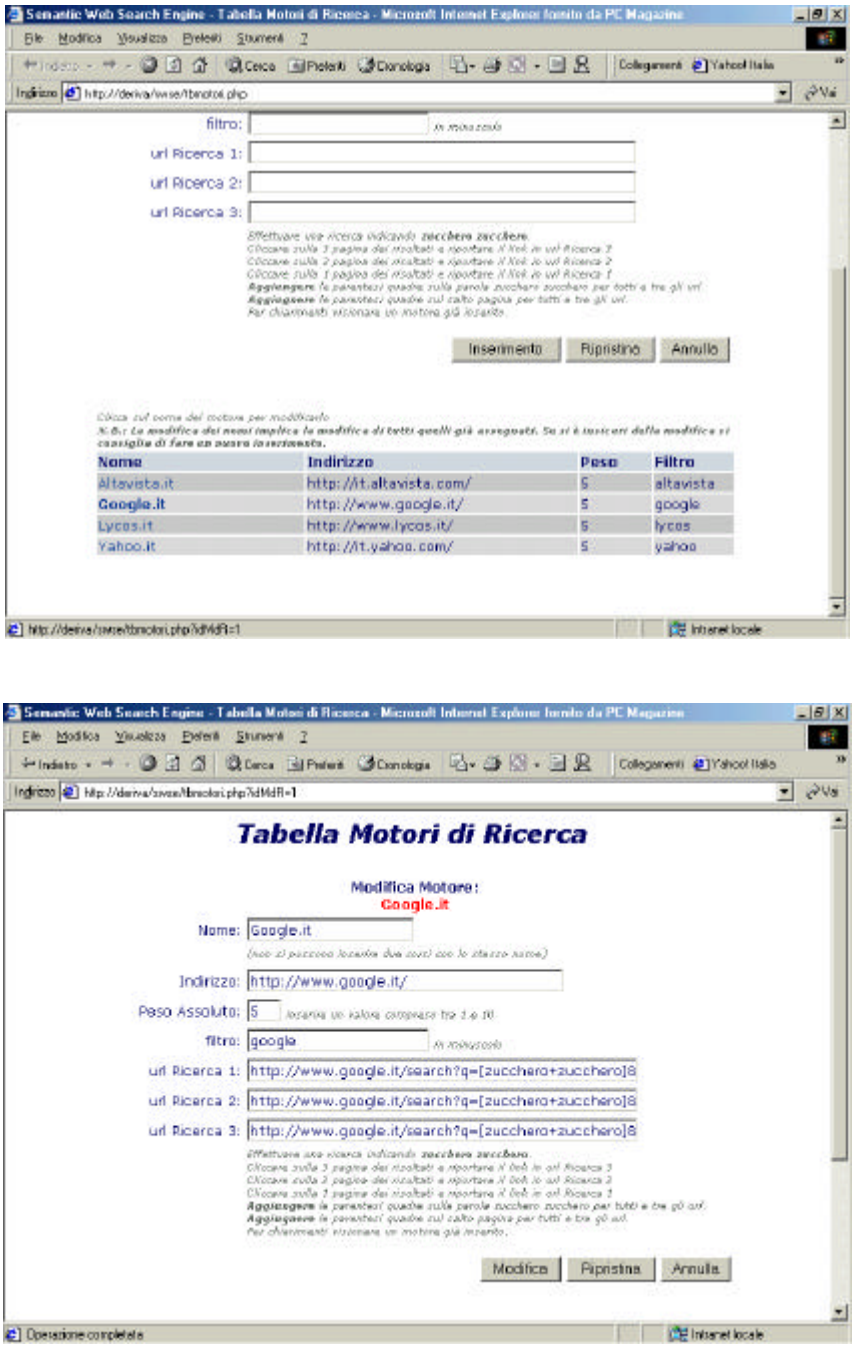

Figura 6.11b: selezione e modifica di un motore di ricerca.

Per effettuare la modifica di un motore di ricerca sarà necessario selezionarlo nell'elenco riportato. La selezione permetterà ad uno script lato server di caricarne i dati relativi e di indicarli all'amministratore nel modulo di modifica. L'amministratore potrà modificarli e chiederne l'aggiornamento: anche in questo caso i dati prima di essere inviati al server verranno controllati localmente.

Capitolo 7

Sperimentazione

7.1 Ambiente utilizzato
7.2 Prove effettuate

Il sistema è stato progettato e sviluppato ponendo molta attenzione alle specifiche: possiamo affermare che sono state totalmente soddisfatte e che il “motore di ricerca semantico” è pronto per essere attivato e reso disponibile agli utenti. Questo si rende necessario per poter constatare il grado di efficacia ed efficienza del sistema in relazione alla metrica utilizzata.

Per poter attivare in maniera corretta il sistema, la Semantic Knowledge Base dovrebbe essere quanto più esaustiva possibile, indicando un numero non eccessivo di domini che dovranno essere non ambigui, non ridondanti e semanticamente validi, ossia con una completa rappresentazione concettuale.

Inoltre, i vari server dovrebbero essere installati ed attivati su macchine differenti con buone connessioni alla rete e alla base dati.

Poiché il sistema è stato sviluppato in maniera estremamente modulare, è possibile prevederne diverse repliche, ciascuna con una propria metrica. In questo modo sarà possibile testare e confrontare l’efficacia delle varie analisi semantiche.

7.1 AMBIENTE UTILIZZATO

La prova finale dei server spider, preprocessor e miner è stata effettuata presso il laboratorio per tesisti del biennio (via Claudio). I server sono stati installati su tre macchine differenti, con le seguenti caratteristiche:

- Fast Ethernet 2.5 Gb;
- S.O. Microsoft Windows XP;
- Java Sun Virtual Machine JDK 1.4.2;
- Sun rmiregistry per la registrazione degli oggetti remoti;

Su una delle tre macchine, precisamente quella occupata dallo spider, sono stati installati i seguenti componenti:

- MySQL versione 3.23.48 alla porta 3306;
- Web Server Apache 1.3.19 alla porta 80;
- PHP 4.0.4 eseguito come modulo aggiuntivo su apache;
- Zend Optimizer 1.0 come ottimizzatore di pagine php;
- Apache Tomcat 4.1 come servlet container, alla porta 8080;
- JK Connector 1.2.5, per usare Apache come web server e creare un ponte con tomcat solo per le servlet ;
- PHPMyAdmin 2.2.4 per la gestione del DBMS;

La servlet, con nome client, è stata inserita nell'applicazione semanticWeb creata sul servlet container Tomcat, congiuntamente agli oggetti remoti.

Il relativo file xml di gestione delle classi dovrà essere opportunamente modificato, puntando agli indirizzi rmi dei relativi server, indicando l'indirizzo IP della relativa macchina:

- Server Spider: rmi://indirizzoIP/
- Server Preprocessor: rmi://indirizzoIP/
- Server Miner: rmi://indirizzoIP/

Inoltre dovrà essere inserito, sempre nel file xml, l'indirizzo relativo alla home page del sito del motore di ricerca (http://serverWeb/swe).

I server del sistema, in fase di attivazione, dovranno comunicare l'indirizzo base per gli oggetti remoti (presenti nell'applicazione semanticWeb di Tomcat)

http://indirizzoIP:8080/semanticWeb/

L'insieme delle pagine web è stato inserito nel web server apache all'indirizzo

http://indirizzoIP/swse/

e lo script php di ricerca, alla fine caricherà la servlet puntando all'indirizzo

http://indirizzoIP:8080/semanticWeb/servlet/client?idQuery=1&nomeDominio=musica

passandogli l'identificativo della query e il nome del dominio (in questo caso 1 e musica).

ATTIVAZIONE DEL SISTEMA

Per attivare correttamente il sistema è necessario avviare prima i vari server di supporto, nel seguente ordine:

1. MySQL;
2. Apache Tomcat;
3. Apache;

Successivamente, per ogni server, attivato su macchine differenti,

```
java
–Djava.rmi.server.codebase=http://indirizzoIP:8080/semanticWeb/
nomedelserver
```

In questo modo verranno avviate le applicazioni server e verranno registrati gli oggetti: il client del sistema (servlet) potrà reperire dinamicamente gli oggetti remoti dall'indirizzo indicato in codebase.

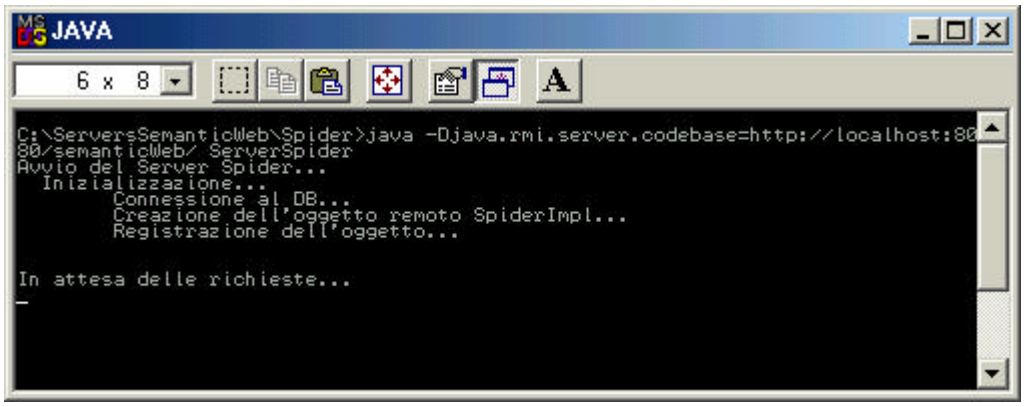

Figura 7.1a: avvio del server spider.

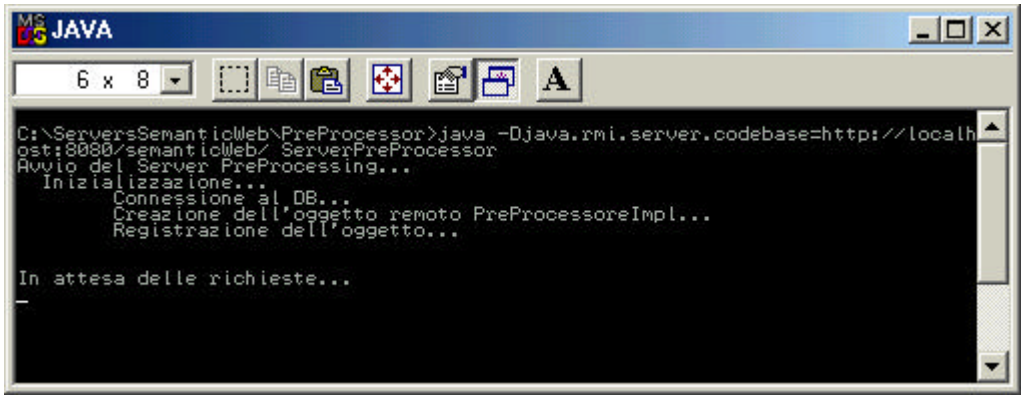

Figura 7.1b: avvio del server preprocessor.

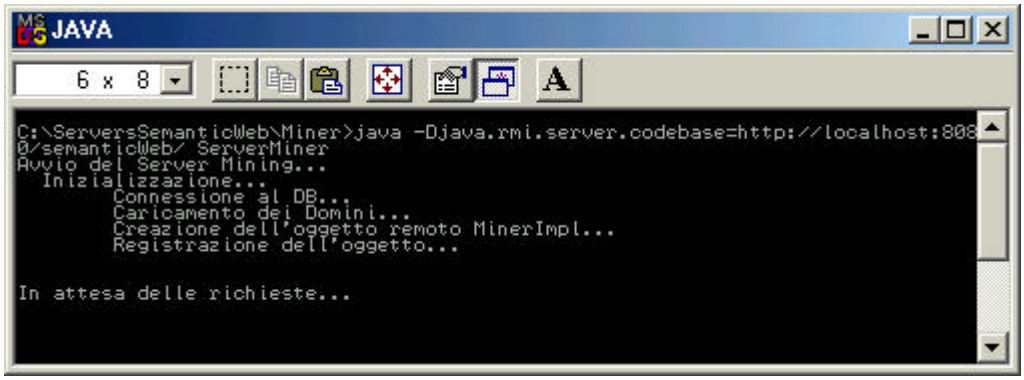

Figura 7.1c: avvio del server spider.

Ora il sistema è pronto per avviare le elaborazioni. E necessario collegarsi con un normale browser al sito http://indirizzoIP/swse/ e inserire una nuova ricerca.

7.2 PROVE EFFETTUATE

Per testare l'efficacia del sistema si sono individuate alcune query ambigue, per le quali i motori di ricerca sintattici restituiscono un set di pagine abbastanza vario.

Supponiamo quindi che l'utente del motore di ricerca semantico voglia ricercare:

- le pagine che parlano del cantante Morandi;
- le pagine che parlano del cantante Zucchero;
- le pagine che parlano della nave scuola "Amerigo Vespucci";
- le pagine che parlano della pesca come sport;

Per le ricerche utilizziamo i parametri di default, ovvero:

- ✓ numero massimo di Link: 20 link per motore;
- ✓ timeout pagine: 1 minuto;
- ✓ Threshold: 0.1;
- ✓ peso Titolo: 30%
- ✓ peos Descrizione: 10%
- ✓ peso KeyWords: 10%
- ✓ peso Corpo: 50%
- ✓ peso SYG: 0.10
- ✓ peso SSG: 5
- ✓ peso SeG: 70

e selezioniamo i seguenti motori di ricerca:

- ✓ Altavista.it, con peso relativo pari al 25%;
- ✓ Google.it, con peso relativo pari al 25%;
- ✓ Yahoo.it, con peso relativo pari al 25%;
- ✓ Lycos.it, con peso relativo pari al 25%;

CONSIDERAZIONI FINALI

Analizzando i risultati di seguito esposti sarà possibile constatare che la ricerca è estremamente efficace: le pagine con voto maggiore sono effettivamente attinenti al dominio cercato. E' importante notare anche l'ordine con cui vengono restituite dai motori di ricerca: come previsto, le pagine semanticamente valide non sono sempre in prima posizione. Per quanto riguarda l'efficienza è importante constatare che il sistema risulta avere ottime prestazioni: ogni ricerca viene conclusa in un tempo inferiore a 1':16'' (nel migliore dei casi 38''). Con operazioni di tunning sui tre server e sulle relative macchine, il valore può ulteriormente diminuire.

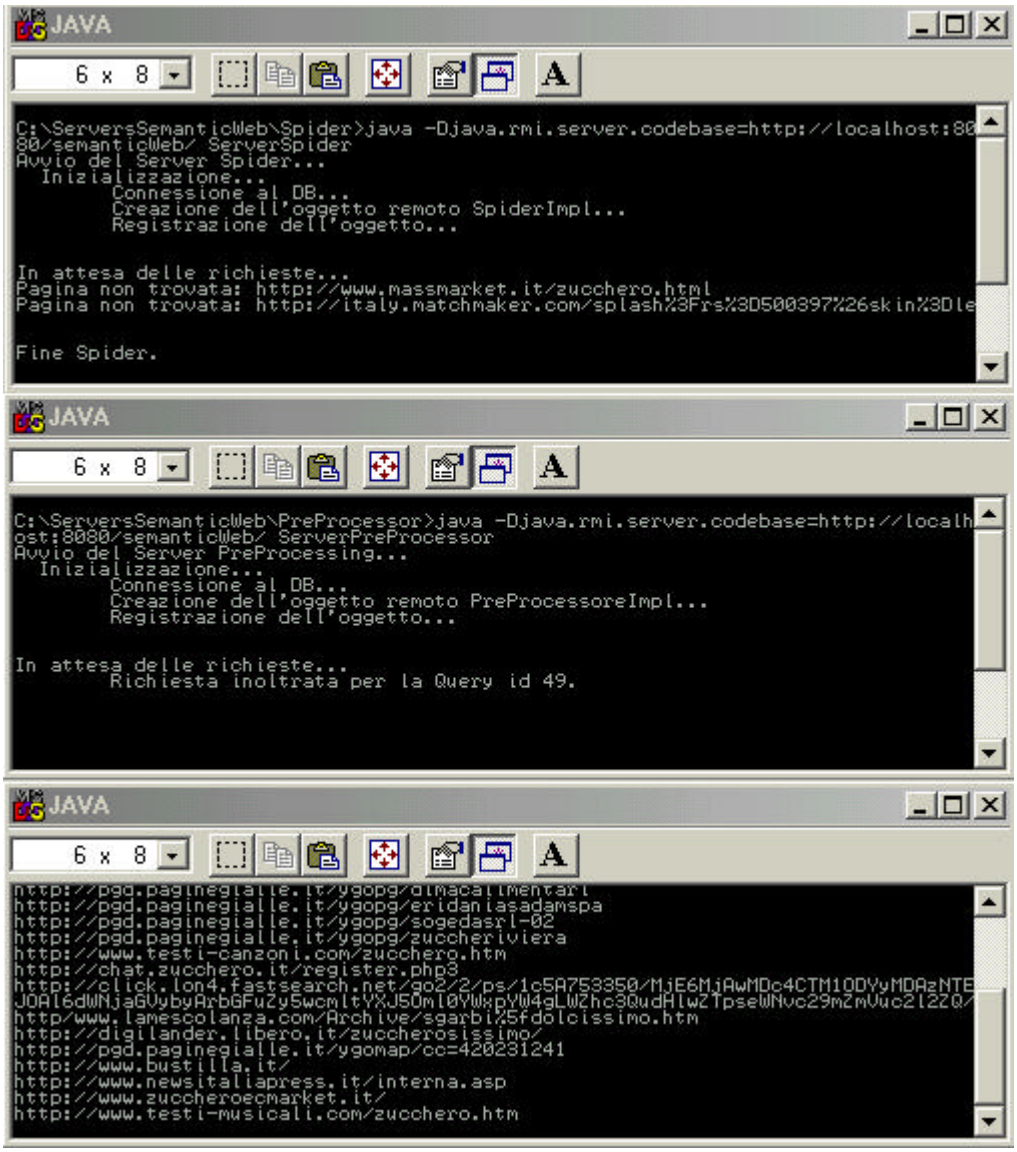

Figura 7.2: elaborazione dei server.

QUERY MORANDI – MUSICA

L'utente vuole le pagine che parlino del cantante Morandi: dovrà compilare il modulo di ricerca indicando morandi e scegliendo il dominio musica.

Tempo di elaborazione: 1':12''

Tempo di Elaborazione 72204 msec.
Lo spider è terminato dopo 64553 msec.
il preprocessing dopo 68299 msec.

Link	**Voto**
http://musicaitaliana.com/linx/giannimorandi.html	0.0883
Posizioni: Lycos.it 16	
http://www.svago.com/morandi.htm	0.0475
Posizioni: Lycos.it 20	
http://digilander.libero.it/comefabeneilgiannino/Come%20fa%2 ...	0.0303
Posizioni: Yahoo.it 14 Google.it 15	
http://rolli.clarence.com/archive/035253.html	0.0155
Posizioni: Google.it 17 Yahoo.it 16	
http://www.clictv.it/spettacolo/varieta/10931.php	0.0125
Posizioni: Google.it 20 Yahoo.it 19	
http://www.morandieditions.com/index_italia.html	0.0039
Posizioni: Altavista.it 4 Google.it 10 Lycos.it 7 Yahoo.it 8	
http://www.exibart.com/IDNotizia1694.htm	0.0027
Posizioni: Lycos.it 19	
http://www.comune.bologna.it/bologna/Cultura/Museicomun/Mora ...	0.0014
Posizioni: Yahoo.it 5 Google.it 6	
http://www.museomorandi.it/	0.0011
Posizioni: Yahoo.it 1 Altavista.it 1 Google.it 1 Lycos.it 2	
http://click.lon4.fastsearch.net/go2/2/ps/1c23975F3D/MiE6MjA ...	0.0005
Posizioni: Lycos.it 10	
http://www.morandimania.it/	0.0004
Posizioni: Altavista.it 6 Google.it 3 Lycos.it 3 Yahoo.it 2	
http://80.17.30.254/Extra/miniSiti/nonapriteguellaporta/game ...	0.0003
Posizioni: Lycos.it 1	
http://www.moranditappeti.it/	0.0003
Posizioni: Yahoo.it 3 Altavista.it 5 Google.it 4 Lycos.it 6	
http://www.museomorandi.it/index_exp.htm	0.0003
Posizioni: Altavista.it 2 Google.it 2	
http://ilgiorno.quotidiano.net/	0.0002
Posizioni: Altavista.it 20	
http://ilrestodelcarlino.quotidiano.net/canali/giovanni_mora ...	0.0002
Posizioni: Google.it 7 Yahoo.it 9	
http://lanazione.quotidiano.net/canali/giovanni_morandi/	0.0002
Posizioni: Google.it 9 Yahoo.it 7	
http://spazioinwind.libero.it/projectmanager/	0.0002

Posizioni: Altavista.it 9 Google.it 5 Yahoo.it 4

http://www.comune.grizzanamorandi.bo.it/ 0.0002

Posizioni: Yahoo.it 6 Altavista.it 15 Google.it 8 Lycos.it 4

http://www.pml-morandi.it/ 0.0002

Posizioni: Yahoo.it 11 Altavista.it 7 Google.it 12 Lycos.it 5

http://space.tin.it/clubnet/hofllp/morandi/ 0.0001

Posizioni: Altavista.it 16 Google.it 19 Yahoo.it 18

http://www.cartaria-morandi.com/ 0.0001

Posizioni: Altavista.it 14 Lycos.it 8

http://www.comune.bologna.it/bologna1/Cultura/Museicomun/Mor ... 0.0001

Posizioni: Altavista.it 3

http://www.giannimorandi.info/ 0.0001

Posizioni: Lycos.it 13 Yahoo.it 10 Altavista.it 12 Google.it 11

http://www.hotelmorandi.com/ 0.0001

Posizioni: Lycos.it 9 Altavista.it 10

http://www.hotelmorandi.it/ 0.0001

Posizioni: Google.it 13 Yahoo.it 13 Altavista.it 8

http://www.italyposters.com/morandi.html 0.0001

Posizioni: Google.it 14 Yahoo.it 12

http://www.marcomorandi.cjb.net/ 0.0001

Posizioni: Altavista.it 11 Google.it 16 Yahoo.it 15

http://www.romagna.net/gmorandi/ 0.0001

Posizioni: Altavista.it 18

http://www.gcmorandi.it/ 0.0000

Posizioni: Lycos.it 18

http://www.guggenheim-venice.it/italiano/06_artisti/morandi. ... 0.0000

Posizioni: Yahoo.it 17 Google.it 18

http://www.marchetti-morandi.com/azienda.htm 0.0000

Posizioni: Lycos.it 12

http://www.morandicarpets.com/ 0.0000

Posizioni: Altavista.it 13

http://www.morandipavimenti.it/ 0.0000

Posizioni: Lycos.it 14

http://www.romagna.net/ecohabitat/morandi.htm 0.0000

Posizioni: Lycos.it 17

http://www.romagna.net/gmorandi/uno.htm 0.0000

Posizioni: Altavista.it 17

http://www.studiomorandi.it/ 0.0000

Posizioni: Lycos.it 15

http://www.welfare.gov.it/ 0.0000

Posizioni: Altavista.it 19

QUERY ZUCCHERO – MUSICA

L'utente vuole le pagine che parlino del cantante zucchero: dovrà compilare il modulo di ricerca indicando zucchero e scegliendo il dominio musica.

Tempo di elaborazione: 1':16''

Tempo di Elaborazione: 76520 msec.
Lo spider è terminato dopo 72144 msec.
il preprocessing dopo 74377 msec.

Link	**Voto**
http://fly.to/zucchero	0.1717
Posizioni: Yahoo.it 2 Google.it 4	
http://www.fly.to/zucchero	0.1715
Posizioni: Lycos.it 15	
http://www.italica.rai.it/principali/argomenti/discografie/z ...	0.1498
Posizioni: Altavista.it 14	
http://www.zucchero.it/english/english.htm	0.1281
Posizioni: Altavista.it 2 Google.it 3	
http://store.zucchero.it/	0.1121
Posizioni: Google.it 13 Yahoo.it 11	
http://www.composizioni.it/zuccheroit.htm	0.1088
Posizioni: Yahoo.it 19 Google.it 22	
http://www.unpodizucchero.net/	0.1066
Posizioni: Yahoo.it 4 Lycos.it 3 Google.it 6	
http://www.geocities.com/sugarcollection	0.0987
Posizioni: Lycos.it 18	
http://pgd.paginegialle.it/ygomap/cc=232463072	0.0809
Posizioni: Yahoo.it 22	
http://pgd.paginegialle.it/ygowww/http://www.sogeda-zucchero ...	0.0805
Posizioni: Yahoo.it 23	
http://www.rockol.it/	0.0753
Posizioni: Altavista.it 12	
http://www.intercom.it/~gluca/sugar/index.html	0.0692
Posizioni: Altavista.it 6	
http://www.abicisac.it/	0.0673
Posizioni: Lycos.it 12	
http://www.testi-canzoni.com/zucchero.htm	0.0669
Posizioni: Yahoo.it 14 Google.it 17	
http://www.testi-musicali.com/zucchero.htm	0.0663
Posizioni: Yahoo.it 15 Google.it 18	
http://musicalstore.caltanet.it/cantautori/zucchero%20parte% ...	0.0656
Posizioni: Google.it 19 Yahoo.it 16	
http://www.milleunoalimenti.it/zucche.htm	0.0601
Posizioni: Altavista.it 20 Yahoo.it 12 Google.it 15	
http://www.bustilla.it/	0.0583

Posizioni: Altavista.it 13 Lycos.it 8
http://www.amicidizuccheromania.it/ 0.0569
Posizioni: Altavista.it 8 Lycos.it 2
http://www.pininpero.com/ 0.0493
Posizioni: Lycos.it 5 Altavista.it 4
http://www.personalzucchero.com/ 0.0459
Posizioni: Google.it 5 Lycos.it 6 Altavista.it 3 Yahoo.it 3
http://www.dolci.it/ 0.0393
Posizioni: Altavista.it 19
http://www.butoszucchero.com/ 0.0314
Posizioni: Altavista.it 9
http://www.nasoegola.com/ita/shop/sugarless.html 0.0309
Posizioni: Altavista.it 7 Lycos.it 13
http://digilander.libero.it/zuccherosissimo/ 0.0272
Posizioni: Lycos.it 4
http://www.italica.rai.it/principali/argomenti/musica/zucche ... 0.0259
Posizioni: Altavista.it 5 Yahoo.it 7 Google.it 9
http://pgd.paginegialle.it/ygopg/sogedasrl-02 0.0241
Posizioni: Yahoo.it 21
http://pgd.paginegialle.it/ygopg/zuccheriviera 0.0196
Posizioni: Yahoo.it 33
http://www.raidue.rai.it/R2_HPprogrammi/0,7371,___84,00.html 0.0192
Posizioni: Yahoo.it 20
http://pgd.paginegialle.it/ygopg/dimacalimentari 0.0174
Posizioni: Yahoo.it 29
http://www.cucinaconme.it/zucchero_vanigliato.htm 0.0168
Posizioni: Altavista.it 16
http://www.disinformazione.it/Zucchero.htm 0.0166
Posizioni: Yahoo.it 13 Lycos.it 16 Google.it 16
http://www.scanner.it/musica/shakesugar1566.php 0.0132
Posizioni: Google.it 21 Yahoo.it 18
http://www.lericettedimamma.com/ricerca_zucchero.html 0.0097
Posizioni: Yahoo.it 8 Google.it 10
http://www.zucchero.it/ 0.0087
Posizioni: Yahoo.it 1 Google.it 2 Altavista.it 1
http://www.risateonline.it/ 0.0075
Posizioni: Altavista.it 15
http://www.crs4.it/~riccardo/Letteratura/Pinocchio/Cap17.htm ... 0.0048
Posizioni: Altavista.it 17
http://zucchero72x.blog.excite.it/ 0.0045
Posizioni: Yahoo.it 10 Google.it 14
http://chat.zucchero.it/register.php3 0.0040
Posizioni: Lycos.it 10
http://www.zuccherosnc.it/ 0.0036
Posizioni: Yahoo.it 9 Lycos.it 17 Google.it 11
http://pgd.paginegialle.it/ygopg/eridaniasadamspa 0.0035
Posizioni: Yahoo.it 25
http://www.cucina-pandizucchero.it/ 0.0027
Posizioni: Lycos.it 21

http://click.lon4.fastsearch.net/go2/2/ps/1c5A753350/MjE6MjA ...	0.0025
Posizioni: Lycos.it 9	
http://www.sicildecor.com/	0.0024
Posizioni: Lycos.it 14	
http://www.museodeldiabete.org/storia06.htm	0.0018
Posizioni: Altavista.it 18	
http://www.buonissimo.org/nonna/zucchero.asp	0.0017
Posizioni: Google.it 8 Yahoo.it 6	
http://www.geocities.com/alex_mrfog/	0.0014
Posizioni: Yahoo.it 5 Google.it 7	
http://www.zuccheroecmarket.it/	0.0014
Posizioni: Altavista.it 11 Lycos.it 7	
http://www.eurekazuccheri.com/	0.0012
Posizioni: Lycos.it 20	
http://www.monodose.it/	0.0012
Posizioni: Lycos.it 19	
http://www.my-content.net/mp3-2/1.htm	0.0005
Posizioni: Google.it 1	
http://www.ideazucchero.com/	0.0004
Posizioni: Altavista.it 10	
http://www.newsitaliapress.it/interna.asp	0.0003
Posizioni: Lycos.it 1	
http://pgd.paginegialle.it/ygomap/cc=420231241	0.0002
Posizioni: Yahoo.it 26	
http://pgd.paginegialle.it/ygocon/ct=55	0.0000
Posizioni: Yahoo.it 24	
http://pgd.paginegialle.it/ygomap/cc=223222402	0.0000
Posizioni: Yahoo.it 30	
http://pgd.paginegialle.it/ygomap/cc=408175432	0.0000
Posizioni: Yahoo.it 34	
http://pgd.paginegialle.it/ygowww/http://www.dimacalimentari ...	0.0000
Posizioni: Yahoo.it 31	
http://pgd.paginegialle.it/ygowww/http://www.eridania.it	0.0000
Posizioni: Yahoo.it 27	

QUERY AMERIGO VESPUCCI – MARE

L'utente vuole le pagine che parlino della nave scuola Amerigo Vespucci della Marina Militare Italia: dovrà compilare il modulo di ricerca indicando Amerigo Vespucci e scegliendo il dominio mare.

Tempo di elaborazione: 42''

Tempo di Elaborazione 42191 msec.
lo spider è terminato dopo 34559 msec.
il preprocessing dopo 38245 msec.

Link	**Voto**
http://foto.blog.excite.it/permalink/11157	0.1376
Posizioni: Yahoo.it 15 Google.it 15	
http://foto.blog.excite.it/permalink/11159	0.0947
Posizioni: Google.it 16	
http://www.istruzione.it/news/2002/prot22056_02.shtml	0.0778
Posizioni: Google.it 17 Yahoo.it 16	
http://www.modellismonavale.com/vespucci.htm	0.0341
Posizioni: Lycos.it 7	
http://www.port-of-rome.com/mare_vivo/vespucci.htm	0.0279
Posizioni: Altavista.it 19	
http://www.port-of-rome.com/mare_vivo/av_comanda.htm	0.0275
Posizioni: Altavista.it 20	
http://www.mamolimodel.com/ita/mm9.htm	0.0247
Posizioni: Lycos.it 15	
http://www.deagostini.it/dea/notizie/amerigo.html	0.0240
Posizioni: Yahoo.it 9 Google.it 10	
http://www.analisidifesa.it/images4/vespucci.htm	0.0036
Posizioni: Lycos.it 20	
http://www.gnd.it/~fabio/Vespucci.htm	0.0025
Posizioni: Lycos.it 19	
http://www.cosedimare.com/vespucci.htm	0.0022
Posizioni: Lycos.it 10	
http://www.giroscopio.com/enciclopedica/amerigovespucci.html	0.0014
Posizioni: Google.it 18 Yahoo.it 13	
http://www.safnet.it/	0.0011
Posizioni: Altavista.it 1 Yahoo.it 1 Lycos.it 2 Google.it 1	
http://www.modellismo-navale.it/A5312/av_pag_01.htm	0.0007
Posizioni: Altavista.it 12	
http://www.cld.it/amerigo.html	0.0005
Posizioni: Altavista.it 3 Google.it 2 Lycos.it 3 Yahoo.it 2	
http://www.f3modellismo.it/Navi/ma741.htm	0.0005
Posizioni: Yahoo.it 10 Google.it 11	
http://www.f3modellismo.it/Navi/ma799.htm	0.0005

Posizioni: Lycos.it 14
http://www.marina.difesa.it/vespucci/campagna02.htm 0.0005
Posizioni: Altavista.it 14
http://www.ipsarvespucci.it/ 0.0004
Posizioni: Lycos.it 4 Google.it 3 Yahoo.it 3 Altavista.it 4
http://www.ipsiamvespucci.it/index1.htm 0.0004
Posizioni: Altavista.it 16 Lycos.it 11
http://www.nauticom.it/ 0.0004
Posizioni: Lycos.it 5 Altavista.it 6
http://olimpodeglidei.blog.excite.it/permalink/26889 0.0002
Posizioni: Google.it 4 Yahoo.it 4
http://utenti.lycos.it/ic1suzzara/vespucci.htm 0.0002
Posizioni: Altavista.it 9 Google.it 8 Yahoo.it 7
http://www.comune.livorno.it/itc/ 0.0002
Posizioni: Altavista.it 7 Google.it 9 Lycos.it 6 Yahoo.it 8
http://www.safnet.it/pagine/index.htm 0.0002
Posizioni: Altavista.it 2
http://www.bncf.firenze.sbn.it/notizie/testi/vespucci.htm 0.0001
Posizioni: Google.it 12 Yahoo.it 11
http://www.giornaletecnologico.it/news/200301/26/3e3271af052 ... 0.0001
Posizioni: Google.it 19 Yahoo.it 17
http://www.interportotoscano.com/ 0.0001
Posizioni: Lycos.it 9 Google.it 13 Altavista.it 17 Yahoo.it 12
http://www.ipsarvespucci.it/pof.htm 0.0001
Posizioni: Altavista.it 5
http://www.is1suzzara.it/sussimatto/vespucci.htm 0.0001
Posizioni: Lycos.it 18
http://www.marina.difesa.it/vespucci/ 0.0001
Posizioni: Google.it 6 Yahoo.it 6
http://www.marina.difesa.it/vespucci/aproposito.htm 0.0001
Posizioni: Google.it 7
http://www.modellismo-navale.it/A5312/av_segnali.html 0.0001
Posizioni: Altavista.it 13
http://www.outlet-spacci.it/pagine/amerigovespucci.htm 0.0001
Posizioni: Yahoo.it 5 Google.it 5
http://galileo.imss.firenze.it/milleanni/cronologia/biografi ... 0.0000
Posizioni: Google.it 20 Yahoo.it 18
http://www.dooyoo.it/product/20301.html 0.0000
Posizioni: Yahoo.it 14 Google.it 14
http://www.firenzespecial.it/images/varie/E32.htm 0.0000
Posizioni: Altavista.it 8
http://www.fol.it/lapulce/orari.htm 0.0000
Posizioni: Lycos.it 16
http://www.interportotoscano.com/bandi.htm 0.0000
Posizioni: Altavista.it 18
http://www.licataonline.it/coopvespucci.htm 0.0000
Posizioni: Lycos.it 12
http://www.marina.difesa.it/vespucci/foto/index.htm 0.0000
Posizioni: Altavista.it 15

http://www.marina.difesa.it/vespucci/index.htm	0.0000
Posizioni: Lycos.it 21	
http://www.mediasoft.it/francobolli/francobolli/219.htm	0.0000
Posizioni: Yahoo.it 19	
http://www.navievelieri.it/vespucci.htm	0.0000
Posizioni: Lycos.it 8	
http://www.raisat.rai.it/	0.0000
Posizioni: Altavista.it 11	
http://www.sosstudents.it/unicom99_prog.htm	0.0000
Posizioni: Lycos.it 13	
http://www.sportalrimini.com/article.php	0.0000
Posizioni: Yahoo.it 20	
http://www.ipsiamvespucci.it/	0.0000
Posizioni: Altavista.it 10	

QUERY PESCA – MARE

L'utente vuole le pagine che parlino della pesca come sattività legata al mare: dovrà compilare il modulo di ricerca indicando pesca e scegliendo il dominio mare.

Tempo di elaborazione: 38''

Tempo di Elaborazione 38175 msec.
lo spider è terminato dopo 31795 msec.
il preprocessing dopo 34219 msec.

Link	**Voto**
http://pesca.puntopartenza.it/	0.0096
Posizioni: Altavista.it 9	
http://www.pescaenautica.it/	0.0084
Posizioni: Lycos.it 4 Yahoo.it 15 Google.it 16 Altavista.it 15	
http://www.legapesca.it/	0.0004
Posizioni: Yahoo.it 3 Google.it 1 Altavista.it 4 Lycos.it 6	
http://www.dalton-trail-lodge.com	0.0003
Posizioni: Altavista.it 1	
http://www.overture.com/d/ITm/ays/mbr/	0.0003
Posizioni: Yahoo.it 1	
http://europa.eu.int/comm/dgs/fisheries/index_it.htm	0.0002
Posizioni: Google.it 3 Yahoo.it 5	
http://www.ebay.it	0.0002
Posizioni: Altavista.it 2	
http://www.infopesca.ch/	0.0002
Posizioni: Altavista.it 8 Google.it 2 Yahoo.it 4	
http://www.pesca.ismea.it/	0.0002

Posizioni: Yahoo.it 7 Google.it 6 Altavista.it 5

http://europa.eu.int/comm/fisheries/policy_it.htm 0.0001

Posizioni: Google.it 4

http://www.agcipesca.it/ 0.0001

Posizioni: Altavista.it 16 Google.it 9 Yahoo.it 12

http://www.anteprima.net/Spinningonline/ 0.0001

Posizioni: Yahoo.it 6 Google.it 5

http://www.apneamagazine.com/ 0.0001

Posizioni: Altavista.it 6 Lycos.it 7

http://www.bellapesca.com/ 0.0001

Posizioni: Yahoo.it 14 Altavista.it 20 Google.it 13

http://www.edai.it/ 0.0001

Posizioni: Yahoo.it 10 Google.it 10

http://www.fipsas.it/ 0.0001

Posizioni: Lycos.it 5 Yahoo.it 19 Altavista.it 3

http://www.geocities.com/Eureka/Park/9546/ 0.0001

Posizioni: Google.it 7 Altavista.it 18 Yahoo.it 8

http://www.hobbypesca.com/ 0.0001

Posizioni: Google.it 8 Lycos.it 24 Yahoo.it 9

http://www.ilnuovo.it/nuovo/foglia/0,1007,192428,00.html 0.0001

Posizioni: Lycos.it 3

http://www.pescainmare.com/ 0.0001

Posizioni: Lycos.it 10 Altavista.it 7

http://www.pescare.com/ 0.0001

Posizioni: Altavista.it 12 Lycos.it 12

http://www.pescareonline.it/ 0.0001

Posizioni: Altavista.it 14 Lycos.it 9

http://www.pianetapesca.it/ 0.0001

Posizioni: Yahoo.it 13 Google.it 12 Lycos.it 25

http://www.pipam.com/ 0.0001

Posizioni: Altavista.it 10 Lycos.it 16

http://www.ticino.com/musmalc/muspesca/muspesca-it.html 0.0001

Posizioni: Yahoo.it 11 Google.it 11

http://digilander.libero.it/maxza/ 0.0000

Posizioni: Yahoo.it 17 Google.it 19

http://web.tiscali.it/mario_web_2000/ 0.0000

Posizioni: Yahoo.it 22

http://www.angelfire.com/al/pasquinuzzia/ 0.0000

Posizioni: Yahoo.it 20

http://www.aziendalex.kataweb.it/HomePageCategory/0,1573,212 ... 0.0000

Posizioni: Yahoo.it 21

http://www.bluefishworld.com/ 0.0000

Posizioni: Lycos.it 17

http://www.edai.it/pescain/index.php 0.0000

Posizioni: Google.it 15

http://www.europalex.kataweb.it/HomePageCategory/0,1623,244, ... 0.0000

Posizioni: Yahoo.it 16 Google.it 18

http://www.federcoopesca.it/ 0.0000

Posizioni: Altavista.it 21

http://www.fiorini.it/ 0.0000
Posizioni: Lycos.it 8
http://www.fishingland.it/ 0.0000
Posizioni: Lycos.it 19
http://www.gaggioso.it/ 0.0000
Posizioni: Lycos.it 13
http://www.gardapesca.com/ 0.0000
Posizioni: Google.it 17 Altavista.it 13
http://www.geocities.com/porluis/fischerpage.htm 0.0000
Posizioni: Altavista.it 19
http://www.geocities.com/SoHo/Square/2593/fischerpage.htm 0.0000
Posizioni: Google.it 14
http://www.ilba.it/ 0.0000
Posizioni: Lycos.it 18
http://www.iniziativapesca.federpesca.it/ 0.0000
Posizioni: Altavista.it 11
http://www.lapescamoscaespinning.it/ 0.0000
Posizioni: Altavista.it 17 Lycos.it 20
http://www.maver.net/ 0.0000
Posizioni: Lycos.it 22
http://www.newsic.it 0.0000
Posizioni: Lycos.it 15
http://www.pescarenet.it/ 0.0000
Posizioni: Lycos.it 21
http://www.planetpescasub.com/ 0.0000
Posizioni: Altavista.it 22 Lycos.it 11
http://www.rod-and-reel.com/ 0.0000
Posizioni: Lycos.it 23
http://www.spidernet.it/pesca/ 0.0000
Posizioni: Yahoo.it 18 Google.it 20

Bibliografia

[1] T. Eiter, T. Lukasiewicz, J. J. Lu, and V. S. Subrahmanian. Probabilistic object bases. Technical report, Technical University of Wien, 2001.

[2] P. Fabriani, M. Missiko_, and P. Velardi. Using text processing techniques to automatically enrich a domain ontology. In FOIS'01, pages 270–284. ACM, 2001.

[3] C.-H. Lee and H.-C. Yang. Text mining of bilingual parallel corpora with a measure of semantic similarity. In Proceedings on IEEE International Conference on Systems, Man, and Cybernetics 2001, pages 470–475. IEEE, 2001.

[4] S. Niremburg and V. Raskin. Ontological sematics, formal ontology and ambiguity. In FOIS'01, pages 151–161. ACM, 2001.

[5] F. Sebastiani. Machine learning in automated text categorization. ACM Computing Surveys (CSUR), 34(1):1–47, 2002.

[6] E. Shek, A. Vellaikal, S. Dao, and B. Perry. Semantic agents for content-based discovery in distributed image libraries. In Proceedings IEEE Workshop on Content-Based Access of Image and Video Libraries, pages 19 –23. IEEE, 1998.

[7] R. Srihari, A. Rao, B. Han, S. Munirathnam, and X. Wu. A model for multimodal information retrieval. In 2000 IEEE International Conference on Multimedia and Expo (ICME 2000), volume 2, pages 701 –704. IEEE, 2000.

[8] L. Weihua. Ontology supported intelligent information agent. In Proceedings on the First Int. IEEE Symp. On Intelligent Systems, pages 383–387. IEEE, 2002.

[9] H. Xu, Y. Mita, and T. Shibata. Intelligent internet search applications based on vlsi associative processors. In Proceedings on Symposium on Applications and the Internet (SAINT2002), pages 230–237. IEEE, 2002.

[10] G. L. Zuniga. Ontology: Its transformation from philosophy to information systems. In FOIS'01, pages 187–197. ACM, 2001.

[11] T. Gruber - "What is an ontology?" - http://ksl-web.stanford.edu/kst/what-is-an-ontology.html

[12] Dizionario della lingua italiana - Centri Garzanti, 1980

[13] C. Latiri, S. Ben Yahia - Generating implicit association rules from Texual Data IEEE

[14] Martin Fowler, Kendall Scott: UML Distilled. Addison Wesley

[15] A. Maciaszek: Sviluppo di sistemi informativi con UML. Addison Wesley

[16] S. Horstmann, Gary Cornell: Java 2 I fondamenti. Mc Graw Hill

[17] S. Horstmann, Gary Cornell: Java 2 Tecniche avanzate. Mc Graw Hill

[18] Jason Hunter e William Crawford: Java Servlet Programming. O'Reilly

[19] Mark Maslakowski: MySQL, guida completa. Apogeo

[20] Tim Converse, Joyce Park: Guida a PHP 4. Mc Graw Hill

[21] David Flanagan: Javascript la guida. O'Reilly

[22] Atzeni, Ceri, Paraboschi, Torlone : Basi di dati. Mc Graw Hill