



Technical Report

TR2.4.12

TITOLO

Realizzazione di uno strumento per il confronto (matching) di ontologie

DRAFT

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------



Indice

Abstract.....	3
Schema Matcher.....	3
1.1 Approccio Strutturale	3
1.1.1 Children Matcher	3
1.1.2 Graph e SubGraph Matcher	4
1.1.3 Maximum Common SubGraph	5
1.2 Approccio Sintattico.....	6
1.2.1 Edit Distance.....	6
1.2.2 Metriche di similarità tra stringhe.....	7
1.3 Approccio Semantico.....	8
1.3.1 Synonym Matcher.....	8
2.1 Analisi e progettazione in ambito Object Oriented	8
2.1.1 Diagrammi dei casi d'uso.....	8
2.1.2 Diagrammi delle classi.....	12
2.1.3 Implementazione in linguaggio Java	15
2.1.3.1 Uso di Swing per la costruzione dell'interfaccia grafica	16
2.1.4 Descrizione delle funzionalità dello strumento	17
2.1.5 Interfaccia grafica	19
Bibliografia.....	24

PROGETTO LC3	Revisione n*	0	Del	----
--------------	--------------	---	-----	------

ABSTRACT

In questo Technical Report viene descritto uno strumento prototipale che permette il matching tra differenti tipologie di schema ed in particolare ha l'obiettivo di definire un mapping semantico tra due ontologie, ai fini della "riconciliazione" tra ontologie o della ricerca di contenuti e servizi basati sulla similarità di ontologie. In esso vengono utilizzate tecniche per l'ontology matching basate su algoritmi strutturali di isomorfismo e similarità tra grafi, su algoritmi sintattici e sull' utilizzo di Wordnet.

SCHEMA MATCHER

1.1 APPROCCIO STRUTTURALE

1.1.1 CHILDREN MATCHER

Questo matcher strutturale è utilizzato in combinazione con un matcher linguistico, nel nostro caso Synonym Matcher e/o String Distance. Esso, basato concettualmente sul Children Matcher [1], determina la similarità tra due elementi interni (non foglie) basandosi sulla combinazione della similarità dei loro figli, i quali possono essere elementi interni o nodi foglia.

La similarità tra i singoli nodi (denominata *element match*) è ottenuta utilizzando i soli matcher linguistici ed è calcolata nel seguente modo:

$$\textit{elementmatch} \begin{cases} 1 & \text{if } (lm1 \vee lm2 = 1) \\ lm2 & \text{if } (0 < lm2 < 1) \wedge (lm1 = 0) \\ 0 & \text{if } (lm1 = lm2 = 0) \end{cases}$$

dove $lm1$ = linguisticmatch1 (Synonym Matcher)

$lm2$ = linguisticmatch2 (String Distance)

Per comprendere invece la misura di similarità strutturale (denominata ChildrenMatch) bisogna introdurre due ulteriori medie: sub-Concept Similarity e sub-Concept Match.

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------

La sub-Concept Similarity (*subConceptSim*) è la media aritmetica degli *elementmatch* di ogni nodo figlio:

$$\text{subConcept Sim} = \frac{\sum_{i=1}^n \text{EM (figlio } i)}{n}$$

dove n = numero dei figli di un nodo

La *sub-Concept Match* (*subConceptMatch*) è il rapporto tra il numero dei nodi figli che hanno *matchato* e il numero totale dei nodi figli:

$$\text{subConcept Match} = \frac{\text{num (matched figli)}}{\text{num (total figli)}}$$

La misura di similarità strutturale è infine calcolata facendo la media geometrica tra *sub-Concept Similarity* e *sub-Concept Match*:

$$\text{NearMatch} = \sqrt{\text{subConceptSim} * \text{subConceptMatch}}$$

1.1.2 GRAPH E SUBGRAPH MATCHER

Questo matcher strutturale è basato sull'algoritmo VF [2,3,4] il quale è un algoritmo di matching deterministico che verifica sia l'isomorfismo tra grafi che quello tra sottografi. L'algoritmo ha validità generale, poiché non sono imposti vincoli alla topologia dei grafi da "matchare", e può sfruttare informazioni semantiche se disponibili.

La versione originale di tale algoritmo prevede un match tra le informazioni semantiche di due nodi puramente sintattico (cioè solo se le stringhe rappresentanti l'informazione sono le stesse). Nel nostro approccio la valutazione della similarità delle informazioni è invece ottenuta tramite l'utilizzo di un matcher linguistico, *Synonym Matcher* e/o *String Distance*.

Inoltre l'algoritmo genera in uscita la differenza tra due sottografi, limitandosi al primo livello, in termini di numero di archi e nodi necessari affinché i sottografi in questione siano isomorfi.

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------

1.1.3 MAXIMUM COMMON SUBGRAPH

Questo matcher strutturale è basato sull'algoritmo di McGregor [], il quale effettua la ricerca del massimo comune sottografo tra due grafi in input.

Questo algoritmo può essere descritto adeguatamente facendo uso di una Rappresentazione tramite Spazio di Stato. Ogni stato s rappresenta un sottografo comune tra i due grafi che è in costruzione. Questo sottografo comune è una parte del massimo comune sottografo che si sta cercando. In ogni stato, una coppia di nodi non ancora analizzata $(n1, n2)$, il primo appartenente al primo grafo e il secondo appartenente al secondo grafo, è selezionata (finché ne esiste una) attraverso la funzione $NextPair(s, n1, n2)$. La coppia di nodi analizzata attraverso la funzione $IsFeasiblePair(s, n1, n2)$ che controlla se è possibile espandere il sottografo comune, rappresentato dallo stato corrente, aggiungendovi questa coppia, così da ottenere un sottografo più grande. Se l'estensione è possibile, allora la funzione $AddPair(n1, n2)$ aggiunge alla soluzione corrente la coppia $(n1, n2)$. Dopo ciò, se lo stato corrente s non è una foglia nell'albero di ricerca, cioè esiste almeno un nodo che si riferisce al primo grafo che non è stato ancora selezionato attraverso la funzione $NextPair$, allora questo nodo viene selezionato ed inizia l'analisi di un nuovo stato. Dopo che il nuovo stato è stato analizzato, viene chiamata una funzione di backtrack, per ripristinare il sottografo comune dello stato precedente e per selezionare un differente stato. Usando questa strategia di ricerca, quando si sceglie un ramo dell'albero di ricerca, esso viene seguito in profondità finché non si incontra una foglia o finché una condizione di pruning si verifica.

Il primo stato è lo stato vuoto, nel quale nessun nodo ha ancora matchato. Uno pseudocodice dell'algoritmo di McGregor è mostrato in Figura 2.10, mentre un'applicazione dello stesso algoritmo è mostrata in Figura 2.11.

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------

```

procedure McGregor_MCS(s)
begin
    while (NextPair(s,n1,n2))
        if (IsFeasiblePair(s,n1,n2)) then
            s' = AddPair(s,n1,n2);
            if (size(s') > CurrentMCSSize) then
                SaveCurrentMCS(s')
                CurrentMCSSize = size(s');
            end if
            if(!LeafOfSearchTree(s') and !PruningCondition(s')) then
                McGregor_MCS(s');
            end if
            BackTrack(s');
        end if
    end while
end procedure
    
```

Figura 2.10: Pseudocodice dell'algoritmo di McGregor

1.2 APPROCCIO SINTATTICO

1.2.1 EDIT DISTANCE

Questo algoritmo calcola la similarità tra stringhe utilizzando l'algoritmo Edit Distance o Levenshtein Distance[5], il quale calcola il numero di *edit operation* (cancellazioni, inserimenti e sostituzioni) da effettuare per trasformare una stringa s nell'altra t .

Ad esempio:

- Se s è "test" e t è "test", allora $LD(s,t) = 0$, perchè non sono necessarie trasformazioni. Le stringhe sono già identiche;
- Se s è "test" e t è "tent", allora $LD(s,t) = 1$, perchè è sufficiente una sostituzione (cambiare la "s" in "n") per trasformare s in t .

Più grande è la Edit Distance, maggiore sarà la differenza tra le stringhe. Nel nostro algoritmo, le stringhe sono dapprima tokenizzate (cioè convertite a Lower Case e depurate dai segni di punteggiatura) e poi stemmatizzate usando l'algoritmo di Porter Stemmer per poi applicare l'algoritmo di Levenshtein.

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------



Il risultato di questo algoritmo è poi normalizzato rispetto alla dimensione della stringa più lunga per ottenere un valore compreso tra 0 ed 1.

1.2.2 METRICHE DI SIMILARITÀ TRA STRINGHE

Questo matcher è utilizzato in alternativa a quello illustrato in precedenza, e permette di calcolare la similarità tra stringhe utilizzando uno dei seguenti algoritmi:

- Hamming distance
- Levenshtein distance
- Needleman-Wunch distance
- Smith-Waterman distance
- Gotoh Distance
- Block distance
- Monge Elkan distance
- Jaro distance metric
- Jaro Winkler
- SoundEx distance metric
- Matching Coefficient
- Dice's Coefficient
- Jaccard Similarity
- Overlap Coefficient
- Euclidean distance
- Cosine similarity

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------

- Variational distance
- Etc...

1.3 APPROCCIO SEMANTICO

1.3.1 SYNONYM MATCHER

Questo algoritmo utilizza il database di WordNet[6] per trovare i sinonimi delle due stringhe oggetto della comparazione, restituendo un valore pari ad 1 nel caso in cui vi sia una sinonimia tra le suddette stringhe, 0 altrimenti.

2.1 ANALISI E PROGETTAZIONE IN AMBITO OBJECT ORIENTED

In tutti i processi produttivi del software, le fasi di analisi e progettazione sono tra le più importanti e costose in termini di tempo. Se affrontate correttamente, riducono al minimo l'attività di manutenzione che, come è noto, risulta essere quella di gran lunga più dispendiosa. L'obiettivo dell'attività di analisi è stabilire che cosa un'applicazione deve fare, mentre la progettazione si occupa di definire come deve essere fatto. Nell'ambito del paradigma Object-Oriented, sono nate diverse metodologie di analisi e progettazione orientata agli oggetti. La più nota e diffusa è rappresentata dall'UML, un linguaggio di modellazione visuale che ha avuto origine dall'integrazione di tre metodologie sviluppate rispettivamente da G. Booch, I. Jacobson e J. Rumbaugh.

Di seguito vengono presentati i diagrammi rappresentativi dell'analisi e della progettazione in base ai quali è stato implementato il software.

2.1.1 DIAGRAMMI DEI CASI D'USO

Le tecniche espone nel precedente capitolo sono state integrate in un sistema di supporto all'utente, secondo i diagrammi di Figura 26 e 27, che rappresentano i cosiddetti *diagrammi dei casi d'uso* dello strumento.

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------

Il diagramma dei casi d'uso ottenuto dall'analisi del software implementato, è stato realizzato per una più comprensibile descrizione delle funzionalità applicative offerte dal tool.

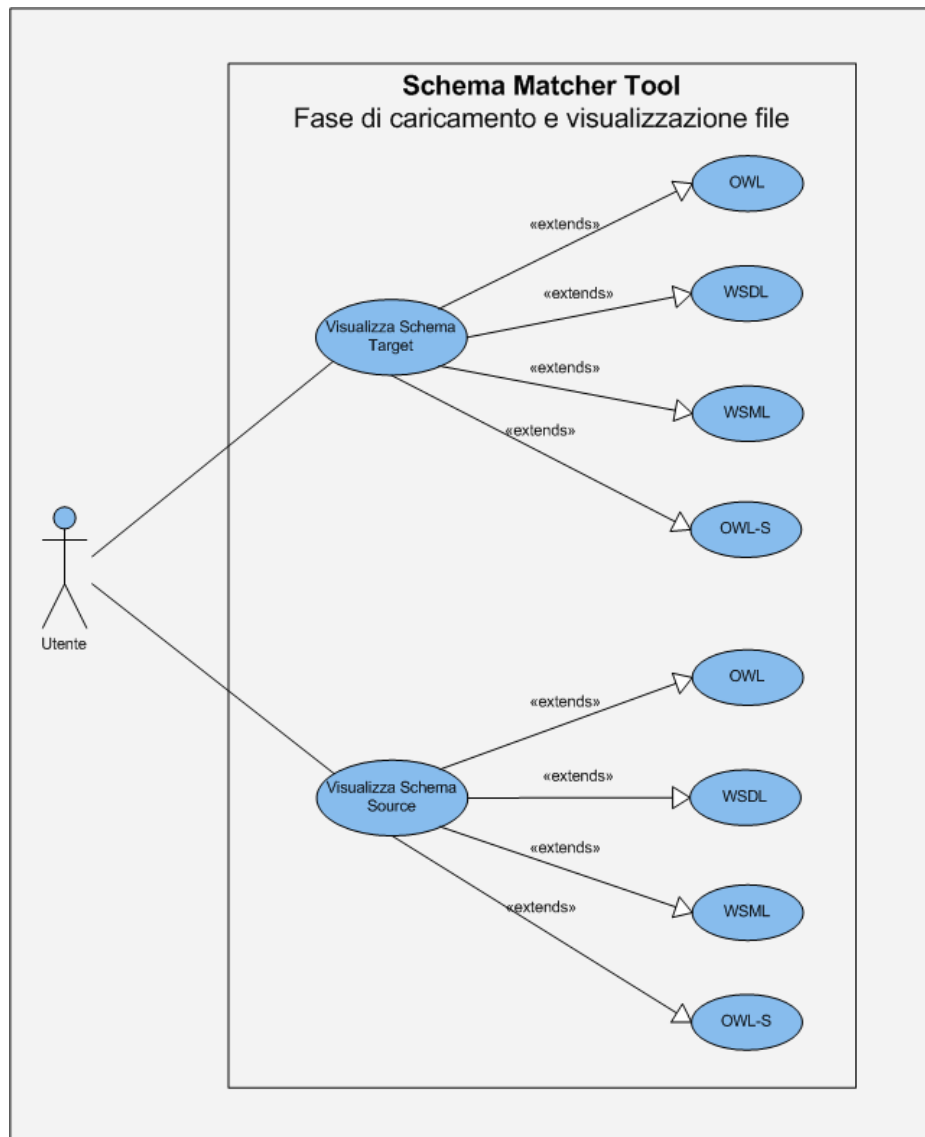


Figura 1: Diagramma dei casi d'uso - Fase di caricamento e visualizzazione

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------

	PARTNER: SECONDA UNIVERSITÀ DI NAPOLI
	RESPONSABILE PROF. BENIAMINO DI MARTINO Technical Report: 2.4.12
<i>LC3 – Laboratorio pubblico-privato di ricerca sul tema della Comunicazione delle Conoscenze Culturali</i>	PAG 10 DI 25

Come si nota dal diagramma in Figura 1 la prima funzionalità offerta dal tool è la visualizzazione di *schema* caricati da file presenti nel file system locale o in remoto tramite URL.

E' inoltre evidente che i diversi tipi di *schema* che possono essere visualizzati sono OWL, WSDL, WSML ed OWL-S, i quali devono essere dapprima analizzati da opportuni parser in java e poi rappresentati mediante grafi diretti radicati, nei quali i nodi e gli archi conservano le informazioni degli *schema*.

In questa prima fase è data inoltre la possibilità di caricare contemporaneamente due *schema* (source e target) in modo da poterne fare un primo confronto prettamente visuale.

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------

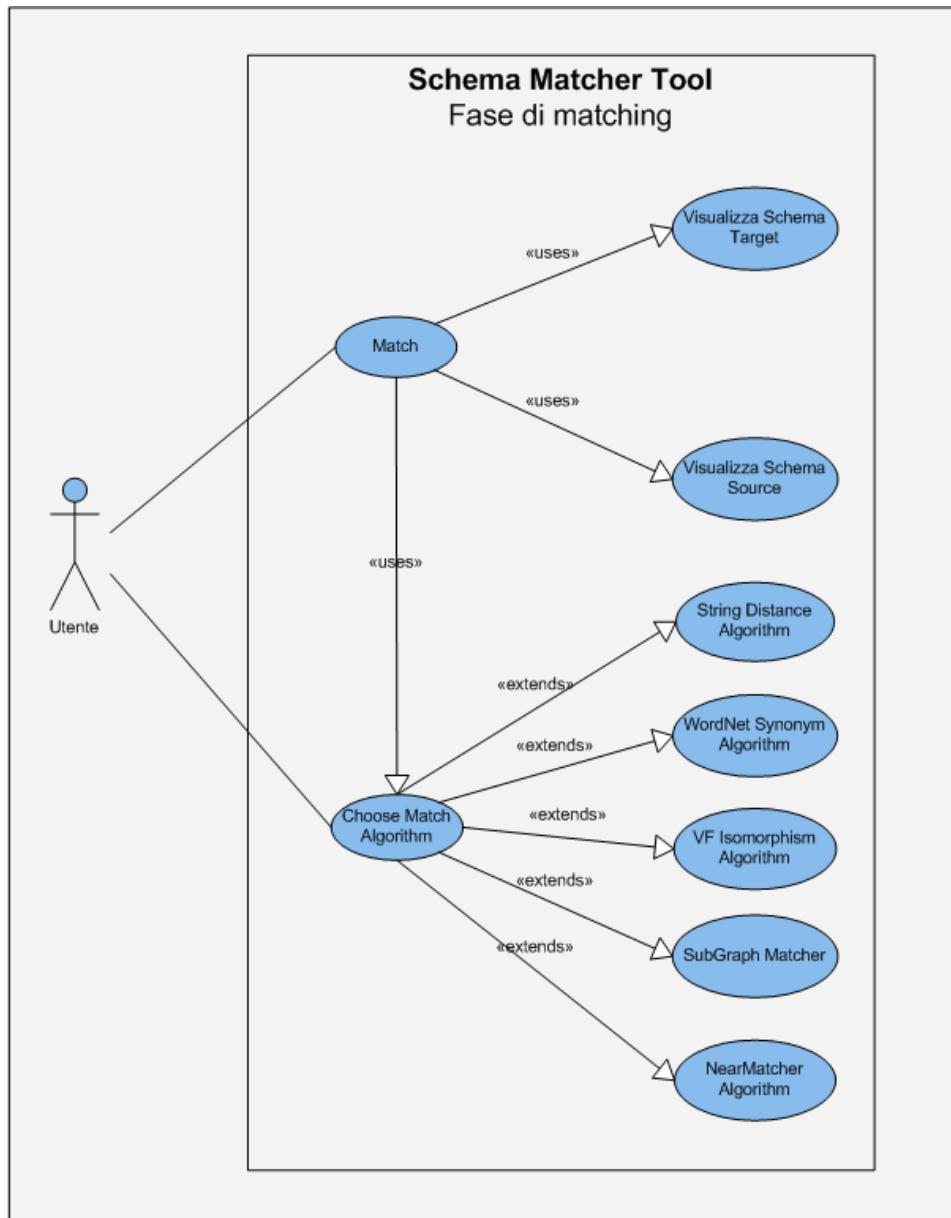


Figura 2: Diagramma dei casi d'uso - Fase di matching

Dal diagramma di Figura 2 si evidenzia la funzionalità principale del tool che è quella del matching strutturale e sintattico di due *schema*.

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------

Tale fase è preceduta dal caricamento dei due *schema* (il cui procedimento è stato descritto in precedenza) e dalla selezione degli algoritmi di Matching.

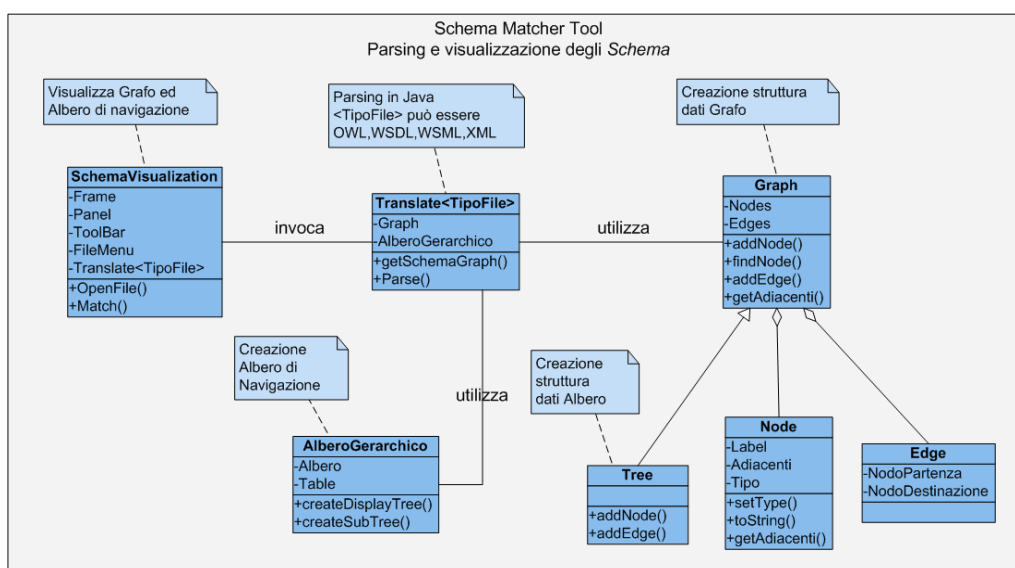
L'utente può scegliere tra cinque diversi algoritmi di match, dei quali tre seguono un approccio strutturale, Children Matcher Algorithm, SubGraph Matcher e VF Isomorphism Algorithm, mentre gli altri due, String Distance Algorithm e WordNet Synonym Algorithm, seguono un approccio sintattico. Viene inoltre data la possibilità di scegliere di utilizzare tali algoritmi separatamente oppure in maniera congiunta.

Una volta effettuata la scelta degli algoritmi di match, l'utente può avviare il vero e proprio processo di Matching, il quale, al termine della computazione, visualizza graficamente i risultati ottenuti.

2.1.2 DIAGRAMMI DELLE CLASSI

Il linguaggio di modellazione UML permette, oltre alla realizzazione del diagramma dei casi d'uso, di mostrare il progetto del software implementato anche in relazione alle singole classi realizzate, attraverso il cosiddetto *diagramma delle classi*.

Si presentano di seguito i diagrammi delle classi relativi alla realizzazione delle scelte applicative presentate dal tool.



PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------

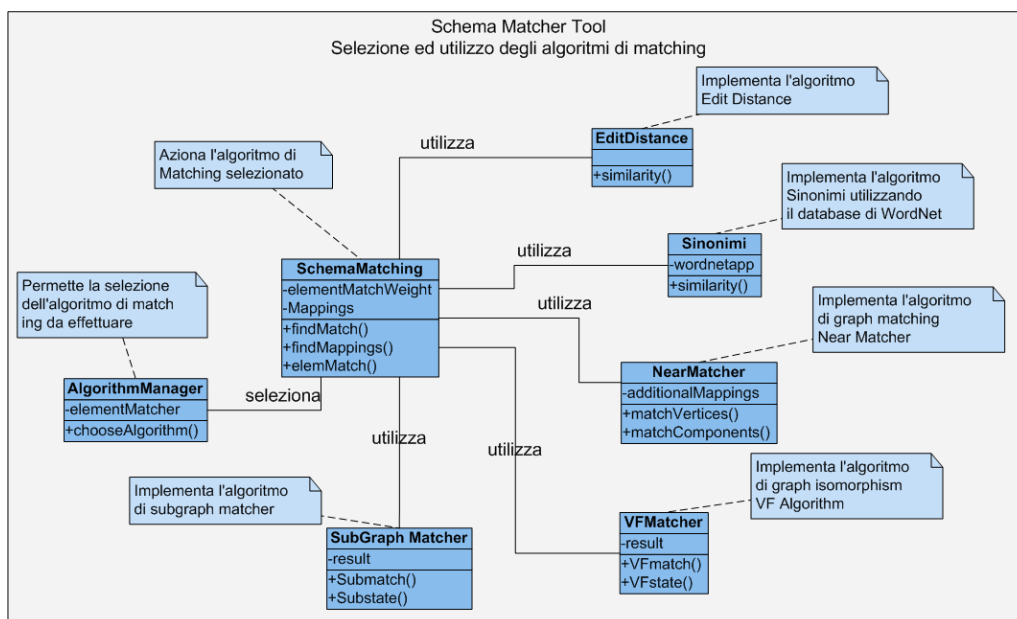
Figura 3: Diagramma delle classi - Parsing e visualizzazione degli schema

Come si nota dal diagramma di Figura 3, la classe *SchemaVisualization* realizza il pannello principale del tool, attraverso il quale è possibile richiamare le funzionalità del tool stesso.

La prima funzionalità richiamabile da *SchemaVisualization* è quella del parsing e della visualizzazione degli *schema* in ingresso. Il parsing è implementato dalle classi *Translate<TipoFile>*, dove *TipoFile* identifica la particolare classe predisposta al parsing del tipo di *schema* scelto (OWL, WSDL, WSML ed OWL-S). Tali classi inoltre memorizzano le informazioni degli *schema* parserizzati nelle strutture dati grafo o albero, le quali sono a loro volta implementate dalla classe *Graph* e dalla sua sottoclasse *Tree*.

La classe *Graph* e la sottoclasse *Tree* provvedono a metodi per l'inserimento di nodi ed archi e per la ricerca degli stessi.

Infine la classe *AlberoGerarchico*, richiamata anch'essa dalle classi *Translate<TipoFile>*, permette l'implementazione di una vista gerarchica degli *schema* analizzati dal tool.



PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------

Figura 4: Diagramma delle classi - Selezione e applicazione algoritmi di matching

Il diagramma di Figura 4, mostra la classe *SchemaMatching* che rappresenta il fulcro del tool sviluppato. Tale classe riceve in ingresso gli algoritmi di match selezionati dall'utente, seleziona ed istanzia le classi implementanti gli algoritmi (*StringDistance*, *Sinonimi*, *ChildrenMatcher*, *SubGraph Matcher* e *VFMatcher*), ed infine memorizza i risultati in un vettore di *Mapping*, il quale sarà visualizzato attraverso la classe *Result* descritta in seguito.

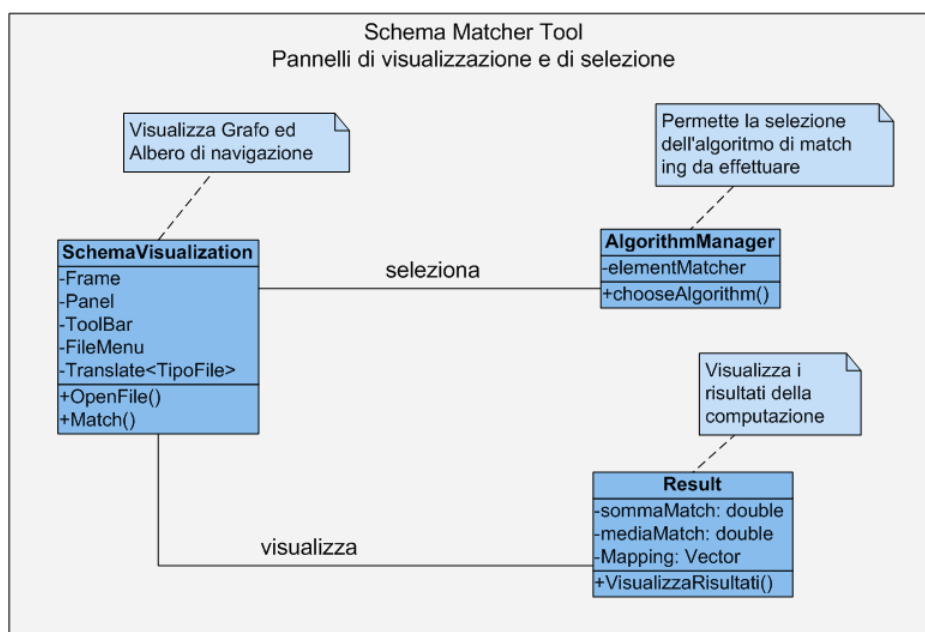


Figura 5: Diagramma delle classi - Pannelli di visualizzazione e di selezione

Il diagramma di Figura 5 mostra le classi che permettono la visualizzazione degli *schema* (*SchemaVisualization*), la gestione degli algoritmi di match (*AlgorithmManager*) e la visualizzazione dei risultati del match (*Result*).

In particolare la classe *Result* contiene il vettore *Mapping*, ricevuto dalla classe *SchemaMatching*, il quale, come già detto, rappresenta il risultato del processo di match.

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------



2.1.3 IMPLEMENTAZIONE IN LINGUAGGIO JAVA

Java è stato presentato come strumento per aggiungere “contenuti dinamici” alle pagine World Wide Web, così che queste non si limitassero più solo a contenere del testo e delle immagini statiche, ma prendessero “vita” grazie a suoni, immagini (anche 3D), animazioni e funzionalità interattive.

Presto Java ha saputo conquistare anche le aziende, soddisfacendo le loro necessità in termini di elaborazione delle informazioni. E’ facile capire, quindi, perché Java ha avuto sin dall’inizio tutte le potenzialità per diventare uno dei linguaggi di programmazione più importanti del mondo.

Java ha saputo eliminare le caratteristiche più complete e inclini agli errori dei linguaggi C/C++ (come i puntatori, i modelli e l’ereditarietà multipla), mantenendo il linguaggio molto conciso ed essenziale. Java è, inoltre, estremamente portabile, e può essere utilizzato per implementare applicazioni basate sul World Wide Web, incorporandovi caratteristiche realmente utili, come le stringhe, la grafica, i componenti GUI, la gestione delle eccezioni, il multi-threading, la multimedialità (suoni, immagini, animazione e video), l’elaborazione dei file e dei database, le reti Client/Server basate sul World Wide Web, l’informatica distribuita e le strutture dati predefinite.

Un altro pregio di Java è dato dall’esistenza in rete di numerosi algoritmi e tool open source sviluppati nel medesimo linguaggio, facilmente reperibili e riutilizzabili. Ad esempio tutti i parser utilizzati e le API di WordNet sono disponibili nel linguaggio Java corredati da una discreta documentazione.

Per tali motivi è stato scelto Java come linguaggio di programmazione per la realizzazione implementativa dello strumento.

Di seguito è inoltre riportato il diagramma delle classi di implementazione, che evidenzia tutte le classi implementate nello strumento prototipale.

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------

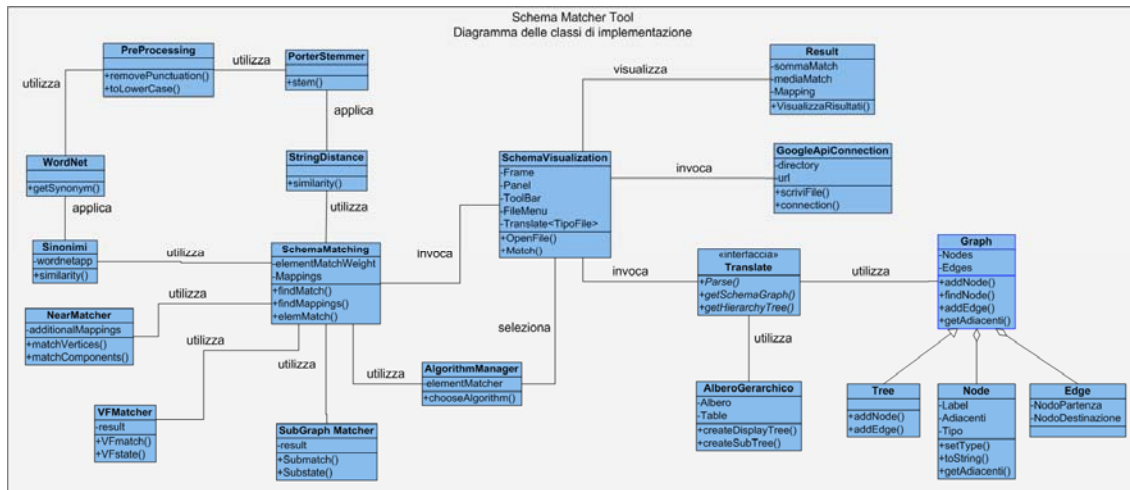


Figura 6: Diagramma delle classi di implementazione

2.1.3.1 Uso di Swing per la costruzione dell'interfaccia grafica

La portabilità di Java consiste essenzialmente nel rendere le applicazioni sviluppate, estendibili in modalità Client/Server; ciò viene ottenuto in quanto Java mette a disposizione due tipologie di programmi, le Applet e le Applicazioni.

Le Applet Java sono programmi proprio pensati per essere inseriti in un documento HTML, trasmesso lungo la rete ed eseguito usando un browser Web; esse sono considerate in pratica un tipo di media che può essere scambiato tramite il Web.

Lo sviluppo di un'interfaccia utente grafica in Java, invece, può essere realizzato utilizzando i componenti della libreria Swing che estende le potenzialità della libreria AWT, lo Standard per la grafica della piattaforma Java2, con le loro proprietà avanzate e alcune caratteristiche fondamentali della GUI, come l'uso dei contenitori e i layout manager.

Nel presente lavoro sono state utilizzate molte funzionalità di tale libreria per rendere l'interfaccia realizzata chiara e interattiva per l'utente.

In particolare sono state utilizzate oltre alla classe `JFrame` per la visualizzazione dei vari pannelli, insieme ai vari componenti in essi contenuti, la classe `JTree` necessaria per la costruzione e la visualizzazione dell'albero gerarchico dei path associati al particolare *schema* analizzato, mentre la rappresentazione grafica della struttura grafo/albero di uno

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------



schema si basa sulla classe `Graph`, una versione adattata e migliorata della demo java `GraphLayout`.

Tali classi fanno anche uso della classe astratta `Graphics` contenuta nel package `AWT` di Java, che permette ad una applicazione di disegnare su componenti definiti in vario modo, come ad esempio immagini off-screen. Un oggetto grafico incapsula varie informazioni supportate da Java come ad esempio le operazioni booleane sui pixel dell'immagine, le operazioni inerenti gli eventi del mouse sull'immagine, impostazione del colore, impostazione dell'area ecc...

E' chiaro, quindi, da quanto detto come Java metta in grado il programmatore di rendere l'applicazione implementata veramente efficace agli scopi che essa stessa si prefigge.

2.1.4 DESCRIZIONE DELLE FUNZIONALITÀ DELLO STRUMENTO

Una volta chiarita la struttura data allo strumento prototipale realizzato, dal punto di vista progettuale ed implementativi, si passa alla descrizione e alla presentazione della sua interfaccia utente.

Viene di seguito riportato un diagramma di flusso che brevemente descrive le fasi di selezione degli *schema*, del parsing e della visualizzazione di detti *schema*, della selezione degli algoritmi di match ed infine della visualizzazione dei risultati del processo di match.

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------

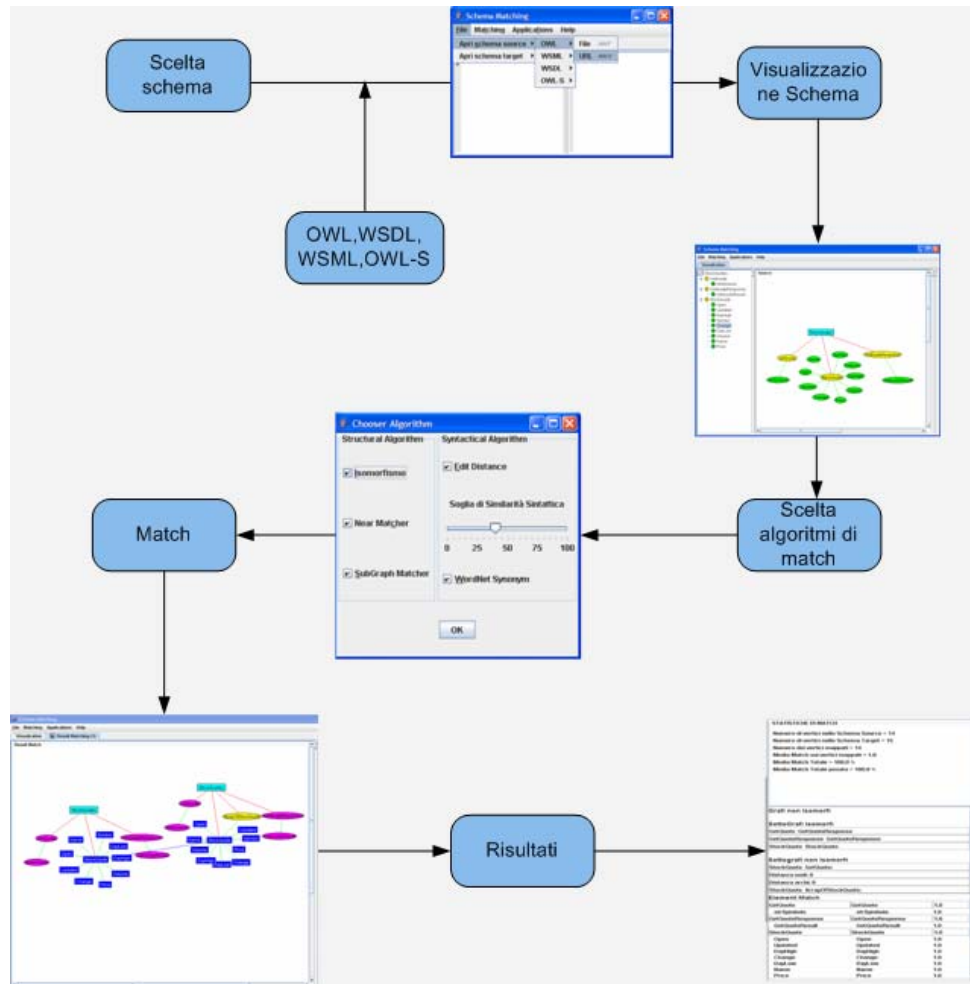


Figura 1: Diagramma di flusso

L'implementazione Java del prototipo ha reso lo strumento semplice dal punto di vista dell'interoperabilità utente, come descritto nel precedente paragrafo.

Di seguito vengono descritti e presentati i pannelli di interfaccia del tool relative a tutte le fasi applicative sopra esposte.

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------

2.1.5 INTERFACCIA GRAFICA

La scelta del linguaggio di programmazione Java, è come detto, anche dovuta alla possibilità di rendere l'interfaccia grafica dello strumento semplice da trattare per l'utente.

L'interfaccia utente presentata, viene riportata in Figura 2.

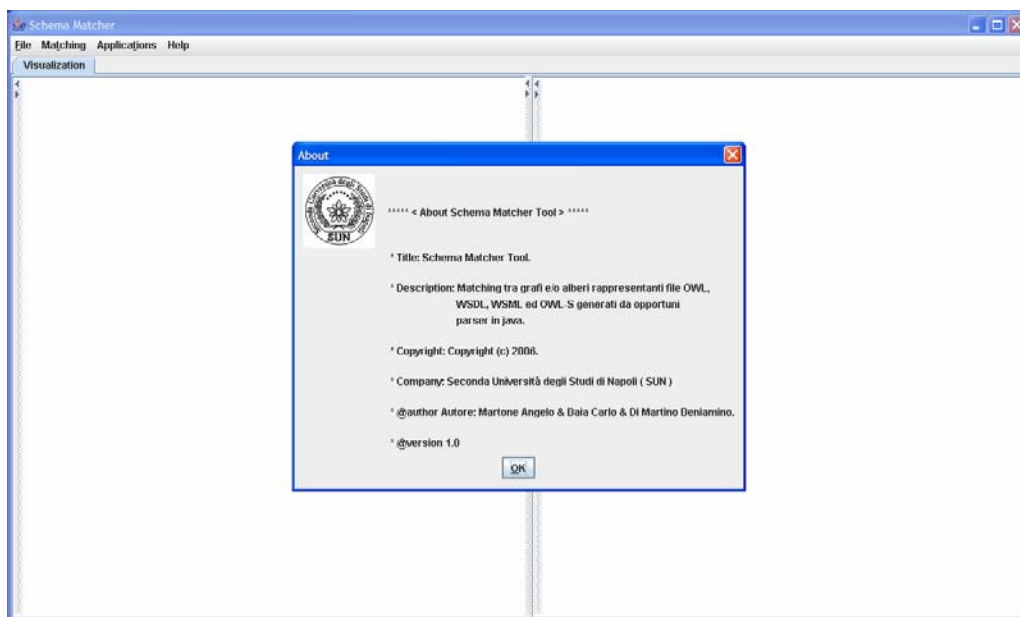


Figura 2: Interfaccia grafica dello Schema Matcher

La fase preliminare da effettuare, come già ampiamente discusso, è rappresentata dal caricamento degli *schema* da analizzare, come mostrato in Figura 34.

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------

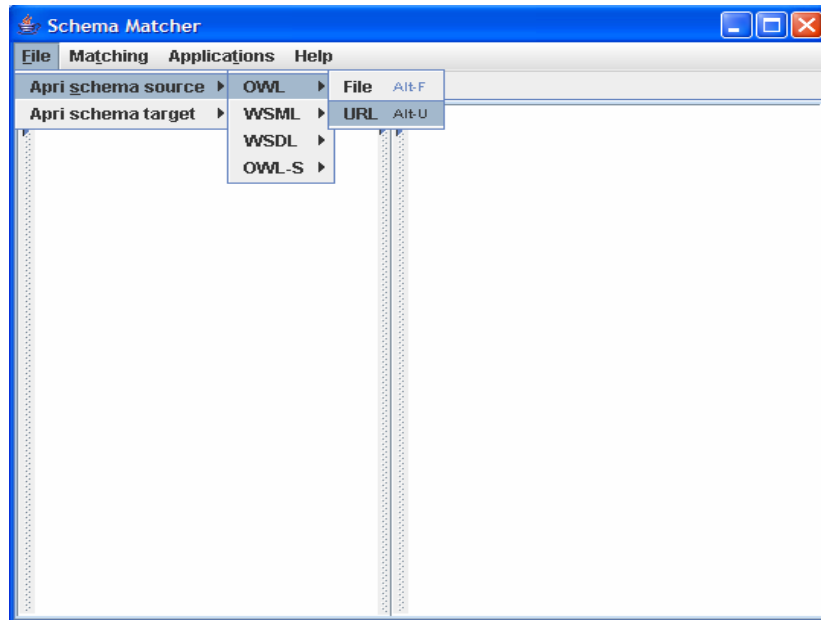


Figura 3: Interfaccia grafica - Selezione file

Dopo la fase di caricamento, vengono visualizzati i grafi corrispondenti agli schema caricati ed i loro alberi di navigazione (alberi gerarchici), Figura 4.

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------

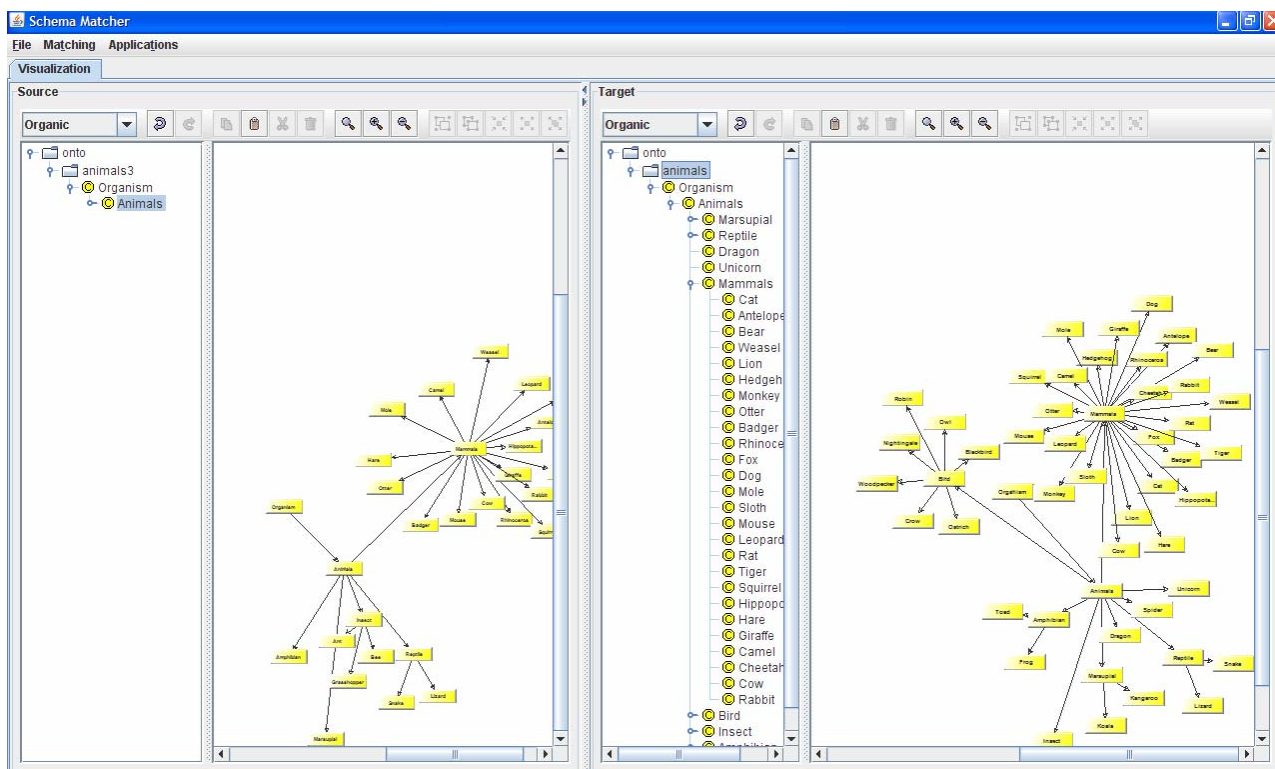


Figura 4 : Visualizzazione delle ontologie da confrontare

Prima di effettuare il processo di matching tra i due *schema*, è possibile scegliere gli opportuni algoritmi, strutturali e sintattici, da applicare. Inoltre è possibile settare, mediante una *slider*, il minimo grado di similarità sintattica intercorrente tra le informazioni contenute all'interno dei nodi dei due *schema* in esame (Figura 5).

PROGETTO LC3	Revisione n*	0	Del	----
--------------	--------------	---	-----	------

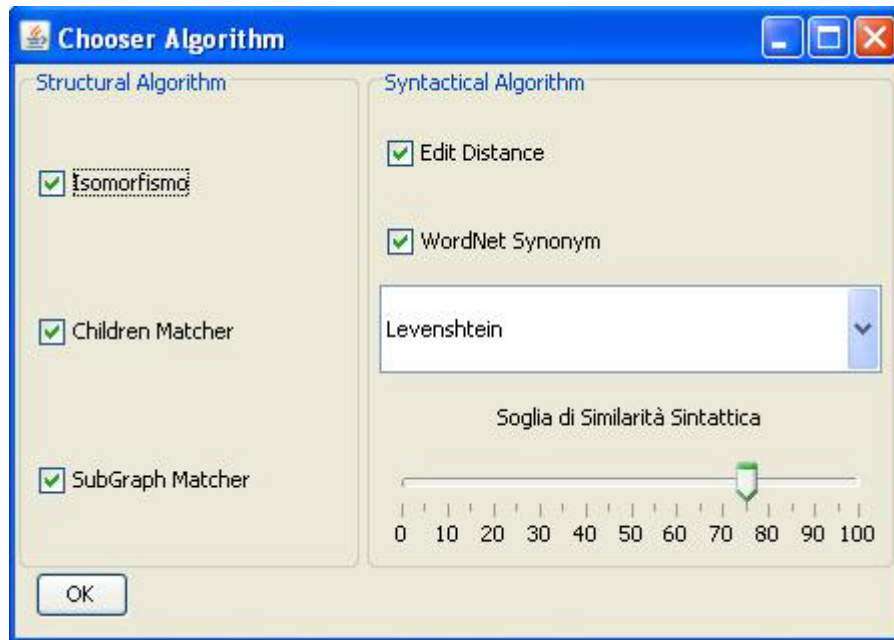


Figura 5: Interfaccia grafica – Scelta algoritmi di matching

Effettuata la scelta degli algoritmi di matching, che di default sono comunque tutti attivi, è possibile avviare il vero e proprio processo di matching. La Figura 37 rappresenta l'interfaccia grafica attraverso la quale vengono visualizzati in viola i nodi dei due *schema* che hanno in “qualche modo” matchato. Per avere una più chiara descrizione riguardante il match di ogni singolo nodo, è necessario cliccare con il tasto destro del mouse sul nodo da esaminare; in tal modo vengono mostrati i nodi che hanno matchato in blue, collegati tra di loro da un ulteriore arco sempre di colore blue, come mostrato in Figura 37. Inoltre vengono visualizzati in blue anche i nodi adiacenti se essi hanno matchato con i nodi adiacenti dell'altro nodo, ciò è indice di un match di tipo strutturale esistente tra i sottografi dei due *schema*.

Nella sezione destra del pannello sono comunque visibili tutte le statistiche ed i risultati di match ottenuti.

Le statistiche di match, Figura 37, riportano le statistiche complessive del match tra i due grafi, ed in particolare:

- numero dei vertici dello *schema* source;

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------

- numero dei vertici dello *schema* target;
- numero dei vertici dello *schema* source mappati sui vertici dello *schema* target;
- media match sintattica dei vertici che hanno matchato;
- media match totale è la percentuale dei vertici dello *schema* source mappati sui vertici dello *schema* target;
- media match pesata è una percentuale ottenuta moltiplicando la media match totale per la media match sintattica.

Lo structural match riporta le statistiche riguardanti il match strutturale tra i due grafi:

- grafi isomorfi o non;
- sottografi isomorfi;
- sottografi non isomorfi e loro distanza in termini di archi e nodi;
- risultato di Children Match.

Infine l'element match riporta le statistiche riguardanti il match linguistico tra due nodi. Ad esempio il valore pari ad 1.0 sta a significare che c'è un match perfetto tra due nodi (la loro Edit Distance è pari a 0 oppure sono sinonimi).

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------

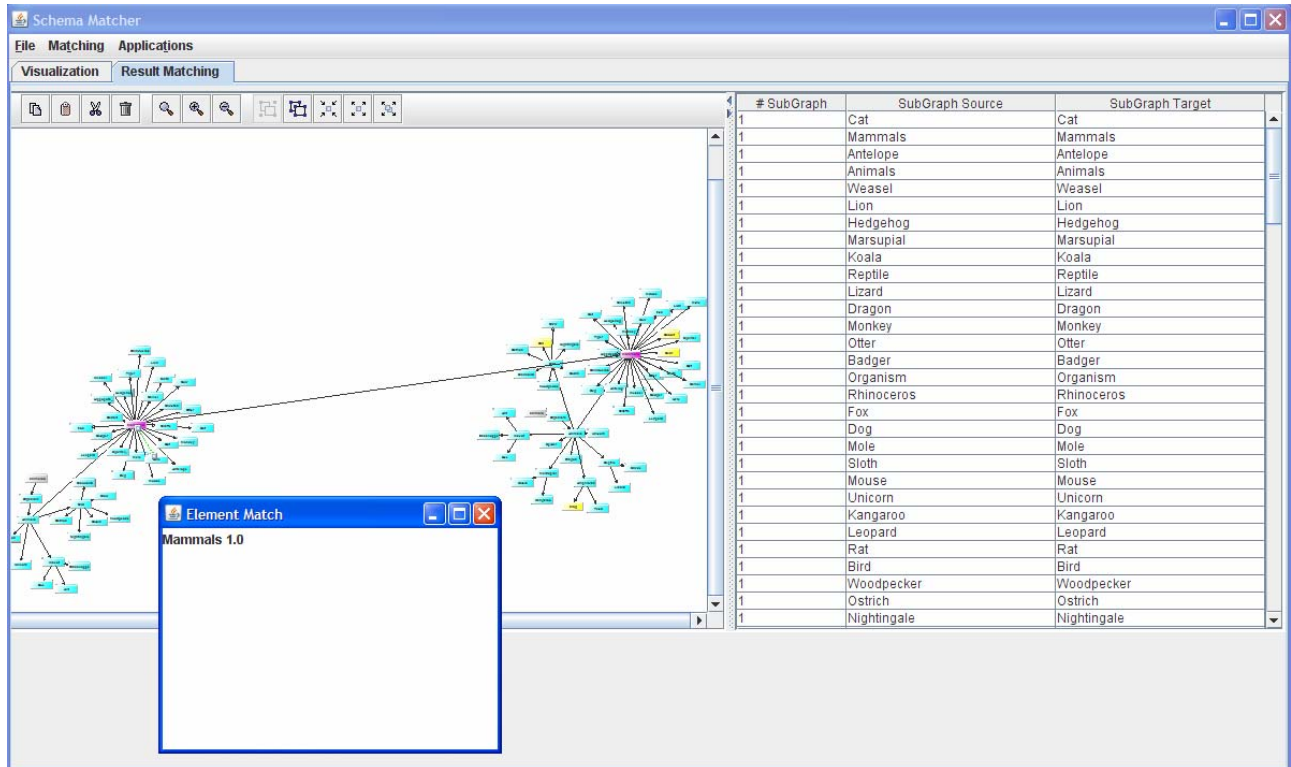


Figura 6 : Visualizzazione dei risultati del matching

BIBLIOGRAFIA

- [1] [COMA - A system for flexible combination of schema matching approaches](#)
- [2] L.P. Cordella, P. Foggia, C. Sansone, M. Vento, [Subgraph Transformations for the Inexact Matching of ARG.pdf](#), *Computing* suppl. 12, pp. 43-52, 1998
- [3] L.P. Cordella, P. Foggia, C. Sansone, M. Vento, [Performance evaluation of the VF Graph Matching Algorithm.pdf](#), Proc. of the 10th ICIAP, IEEE Computer Society Press, pp. 1172-1177, 1999
- [4] L.P. Cordella, P. Foggia, C. Sansone, M. Vento, [Fast Graph Matching for Detecting CAD Image Components.pdf](#), Proc. of the 15th Int. Conf. on Pattern Recognition, IEEE Computer Society Press, vol. 2, pp. 1038-1041, 2000

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------



- [5] M. Gilleland, Merriam Park Software, *Levenshtein Distance Algorithm*, <http://www.merriampark.com/ld.htm>
- [6] Princeton University, “*Wordnet a lexical database for the English language*” - <http://wordnet.princeton.edu>

PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------