



**PARTNER: SECONDA UNIVERSITÀ DI NAPOLI**

**RESPONSABILE PROF. BENIAMINO DI MARTINO**

Technical Report: **2.4.9**

*LC3 – Laboratorio pubblico-privato di ricerca  
sul tema della Comunicazione delle Conoscenze Culturali*

**PAG 1 DI 37**

## **Technical Report**

**TR2.4.9**

**TITOLO**

**Definizione di una architettura per il Semantic Information Retrieval**

<i>PROGETTO LC3</i>	<i>Revisione n*</i>	<i>0</i>	<i>Del</i>	<i>-----</i>
---------------------	---------------------	----------	------------	--------------



## Indice

Abstract.....	4
1.1 Definizione di una architettura per il Semantic Information Retrieval.....	5
1.2.1 User Interface.....	8
1.2.2 Ontology Manager.....	8
1.2.3 Syntactical Parser.....	9
1.2.4 Triple Extractor.....	11
1.2.4.1 Heuristic Pattern Detection.....	12
1.2.5 Semantic Mapper.....	17
1.2.5.1 Subgraph Extraction.....	18
1.2.6 Target and Modifier Extractor.....	22
1.2.7 Visual Query Builder.....	23
1.2.7.1 Notazione Grafica.....	23
1.2.7.2 Triple Pattern Base.....	24
1.2.7.4 Triple Patterns Multipli.....	25
1.2.7.5 Graph Pattern.....	26
1.2.7.6 Optional Graph Pattern.....	26
1.2.7.7 Union Graph Pattern.....	27
1.2.7.8 And Graph Pattern.....	28
1.2.7.9 Ordinamento dei risultati.....	29
1.2.7.10 Filtraggio delle variabili.....	30
1.2.7.11 Distinct, Limit e Offset.....	31
1.2.8 Ontology Querying.....	31
1.2.9 Graph Matching Querying.....	32
1.2.9.1 Matching sintattico e semantico.....	33

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



1.2.9.2	Matching strutturale .....	34
1.2.10	Document Retrieval and Presentation .....	36
1.2.11	Flusso delle fasi elaborative .....	36

<i>PROGETTO LC3</i>	<i>Revisione n*</i>	<i>0</i>	<i>Del</i>	<i>----</i>
---------------------	---------------------	----------	------------	-------------



## ABSTRACT

*In questo Technical Report si definisce una architettura per il Semantic Information Retrieval.*

*Il bisogno di rendere i contenuti del Semantic Web accessibili agli utenti, sta diventando sempre più pressante dal momento che l'ammontare delle informazioni immagazzinate nelle basi di conoscenza di tipo ontologico e le annotazioni semantiche basate su ontologie, aumentano costantemente. Le ontologie, che rappresentano la tassonomia e le relazioni che intercorrono all'interno di un dato dominio di interesse (tramite classi e proprietà) e che ne conservano la conoscenza (tramite le istanze e le relazioni tra di esse), giocano un ruolo fondamentale nel Semantic Web, permettendo lo scambio e la condivisione della conoscenza.*

*Il sistema definito in questo Technical Report, oltre ad utilizzare le tecniche di Information Retrieval e Natural Language Processing, si basa anche sulle ontologie, ma diversamente dagli approcci che tendono ad utilizzarle come un mero supporto al query expansion, l'approccio sviluppato le utilizza per aiutare il sistema nel processamento della query utente fornendo una base di conoscenza specifica che permetta di esplorarne il suo significato semantico e fornisca una risposta precisa a query anche molto complesse.*

*In questo lavoro si è implementata una architettura che è portabile, indipendente dal dominio e che mostra delle buone performance, la quale offre agli utenti la possibilità di ottenere le informazioni di cui hanno bisogno da basi di conoscenza ontologiche.*

*Il gap semantico tra il Semantic Web, basato sulla logica, e il mondo reale degli utenti è superato grazie all'utilizzo di una interfaccia basata sul linguaggio naturale (NLI, Natural Language Interface) che fornisce un mezzo per interrogare le ontologie da parte di utenti non esperti, senza che essi conoscano linguaggi quali RDF, OWL e SPARQL, o che conoscano lo schema ed il vocabolario dell'ontologia.*

*Il sistema sviluppato funge inoltre da motore di Question Answering in quanto le istanze ontologiche ritornate dal processo di retrieval rappresentano la risposta alla domanda posta in linguaggio naturale dall'utente.*

*Esistono tre grandi ostacoli alla realizzazione di un sistema che interpreti le query utente in linguaggio naturale e le mappi in una grammatica formale quale quella delle ontologie.*

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



*Il primo è dato dalla complessità e dall'ambiguità intrinseca del linguaggio naturale che lo rende particolarmente difficile da comprendere da parte di una macchina. La comunità NLP ha svolto molto lavoro in questa area. I parser statistici più avanzati possono raggiungere circa il 90% per quanto riguarda la precision ed il recall. Pur tuttavia è ancora un problema aperto e lontano dall'essere risolto.*

*Il secondo problema è che anche assumendo che la query in linguaggio naturale sia analizzata correttamente dal parser, restano molte difficoltà nella sua effettiva traduzione in una query formale. Il vocabolario della base di conoscenza è più controllato e più snello rispetto a quello dell'utente, cosicché è arduo mappare correttamente i vocaboli dell'utente in quelli della base di conoscenza. Inoltre rappresentazioni della conoscenza diverse richiedono tecniche differenti per interpretare la semantica delle query utente. Ad esempio, come trattare le negazioni presenti nelle query è diverso nel caso delle ontologie rispetto al caso dei database.*

*Un ulteriore problema è come associare le istanze dell'ontologia ai documenti veri e propri che rappresentano il vero obiettivo della ricerca utente.*

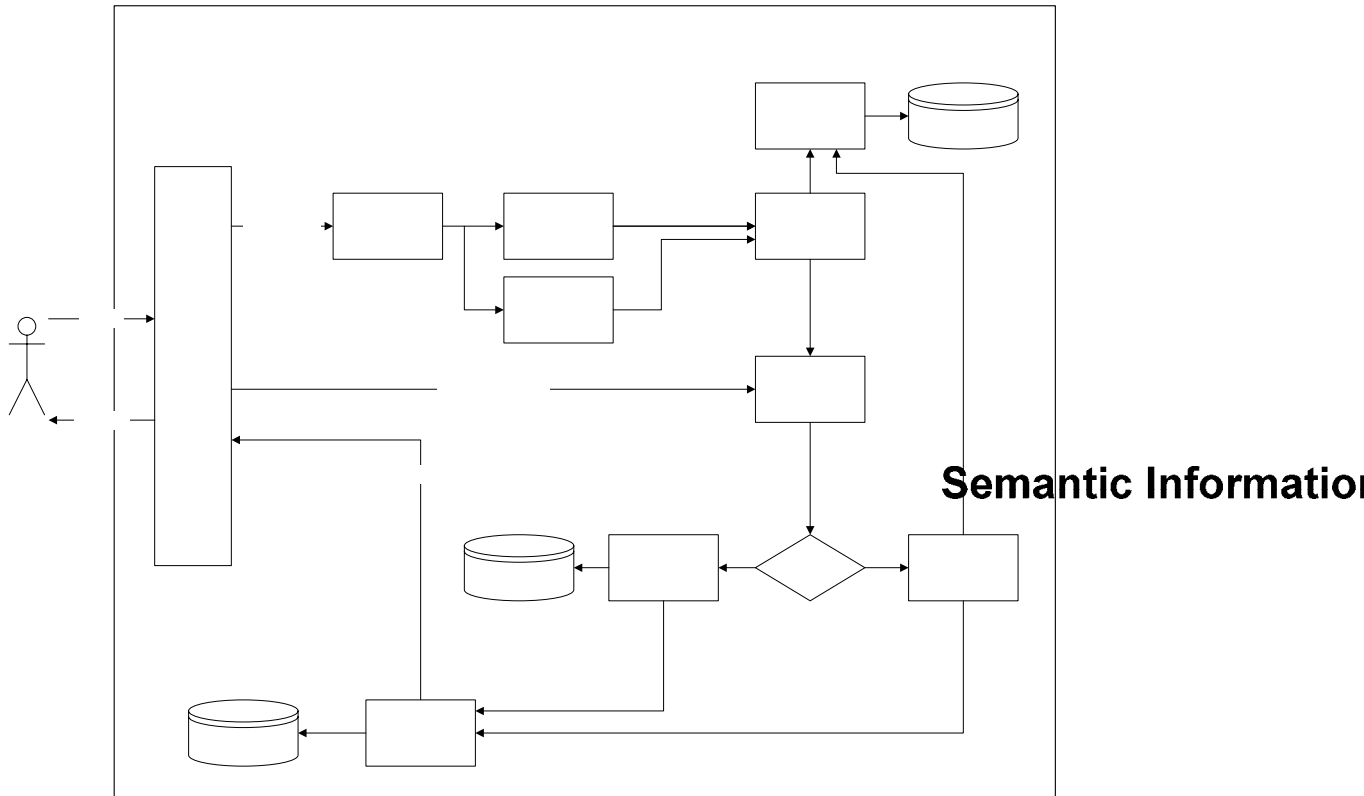
*L'architettura realizzata cerca di risolvere i primi due problemi succitati.*

*Il primo problema è affrontato tramite un parser grammaticale e sintattico mentre il secondo è affrontato utilizzando tecniche di similarità sintattiche e semantiche, tecniche di schema matching e tramite l'utilizzo di euristiche.*

*Infine il terzo problema è affrontato utilizzando l'architettura definita nel [TR2.3.3](#), sotto due aspetti: da un lato si è sviluppato uno strumento che permetta l'annotazione manuale di contenuti multimediali tramite un modello di annotazione semantica appositamente sviluppato, dall'altro lato si è implementata una architettura software che permetta di salvare in modo persistente le annotazioni (e di gestirne le versioni) e, tramite l'utilizzo di un indice, di mantenere i collegamenti tra istanze ontologiche e documenti a cui si riferiscono.*

## **1.1 DEFINIZIONE DI UNA ARCHITETTURA PER IL SEMANTIC INFORMATION RETRIEVAL**

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



**Figura 1.1:** Architettura per il Semantic Information Retrieval

L'architettura sviluppata consiste di diverse componenti ognuna delle quali assolve ad una specifica funzionalità, di seguito si elencano queste componenti con una breve descrizione:

- **User Interface:** l'interfaccia grafica guida l'utente durante tutte le fasi coinvolte nel retrieval semantico; permette all'utente di inserire la query in linguaggio naturale, di validarla successivamente, di navigare i risultati ottenuti e di visualizzare le parti dei documenti annotati;
- **Ontology Manager:** compito di questo modulo è quello di mantenere la persistenza delle ontologie presenti nel sistema e di indicizzarle per permetterne un rapido utilizzo durante le fasi successive;



- [Syntactical Parser](#): tale componente esegue l'analisi lessicale e sintattica della query utente;
- [Triple Extractor](#): tale componente ottiene l'albero sintattico della query, ne estrae uno scheletro formato da tutti i sintagmi riconosciuti in essa, ne ricava tutti i sintagmi nominali e ne ricerca le relazioni esistenti tra ogni coppia di essi. Queste triple sono individuate tramite una ricerca di pattern all'interno della query;
- [Semantic Mapper](#): a questo componente del sistema è affidato il compito di mappare le triple estratte dalla query in linguaggio naturale sulle triple definite nelle ontologie;
- [Target and Modifier Extractor](#): in questa fase si estraggono i *Target* ed i modificatori dalla query in linguaggio naturale;
- [Visual Query Builder](#): in questa fase vengono presentati all'utente i risultati della fase di mapping, cioè uno o più sottografi estratti dalle ontologie. L'utente può direttamente sottoporre la query allo SPARQL Engine oppure validarla andando ad aggiungere/eliminare classi e/o relazioni;
- [Ontology Querying](#): una volta formulata la query questa può essere sottoposta ad un motore di query semantico, a tale scopo sono stati scelti il motore di query ARQ di Jena ed il motore di query del reasoner Pellet, in entrambi i casi il grafo rappresentante la query deve essere parserizzato nel linguaggio SPARQL;
- [Graph Matching Querying](#): la query formulata può essere sottoposta anche ad un motore di matching semantico appositamente sviluppato, il quale cerca le corrispondenze tra il grafo rappresentante la query (di seguito chiamato Query Graph) e i grafi delle annotazioni presenti nel repository delle annotazioni (di seguito denominati Annotation Graph);
- [Document Retrieval and Presentation](#): compito di questa fase è quello di mostrare graficamente agli utenti i risultati ritornati dalla query utente. I risultati sono raggruppati in forma tabellare e rappresentano le istanze definite all'interno dell'ontologia e/o nelle annotazioni che soddisfano i criteri della query.

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



### 1.2.1 USER INTERFACE

L'interfaccia utente permette agli utenti di inserire delle query utilizzando il linguaggio naturale senza restrizioni. Una volta formulata la query, questa viene visualizzata tramite l'interfaccia grafica sottoforma di un grafo. L'utente può, tramite l'interfaccia, validare e/o modificare la query generata, navigando all'interno dell'ontologia e aggiungendo concetti o relazioni alla query stessa.

Terminata questa fase, è possibile verificare la corrispondente sintassi testuale SPARQL generata in automatico che, anche in questo caso, è possibile validare e modificare.

Una volta sottoposta la query al sistema, l'utente può fare il browsing dei risultati, visionare i documenti rilevanti e le parti annotate.

L'interfaccia grafica permette anche di visionare i risultati di matching parziale che sono previsti all'interno dell'architettura.

### 1.2.2 ONTOLOGY MANAGER

Compito dell'*Ontology Manager* è quello di mantenere la persistenza delle ontologie presenti nel sistema e di indicizzarle per permetterne un rapido utilizzo nelle fasi successive. Le ontologie vengono parserizzate tramite il framework Jena e vengono poi indicizzate, in base ai concetti e alle proprietà presenti in esse. Sia le query che le ontologie sono analizzate tramite tecniche di stemming e stopwords elimination. Le etichette delle entità presenti nelle ontologie (proprietà e classi) vengono poi associate anche ai propri sinonimi utilizzando WordNet.

Una volta che l'utente ha inserito una query in linguaggio naturale, tale componente si occupa di ritornare le ontologie più attinenti alla query formulata, in questo modo il numero di confronti effettuati durante la fase di mapping tra la query utente e le ontologie, viene limitato, consentendo una migliore scalabilità del sistema. Infatti essendo il sistema progettato portabile ed indipendente dal dominio, le ontologie gestite sono le più eterogenee possibile e quindi vanno escluse a priori quelle che non hanno alcuna attinenza (per quanto riguarda la terminologia utilizzata) con la query utente.

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

### 1.2.3 SYNTACTICAL PARSER

Scopo di questo componente è quello di parserizzare la query in linguaggio naturale, utilizzando l'analisi grammaticale e sintattica, e di generare il corrispondente albero sintattico.

La tipologia di grammatica adottata per lo sviluppo dell'analisi grammaticale e sintattica è la Context-Free Grammar (CFG), che rappresenta un metodo particolare per la descrizione della sintassi di un linguaggio. La CFG può essere usata per descrivere qualsiasi linguaggio, naturale o artificiale, che rispetti certi vincoli; tali vincoli sono relativi a come le clausole sono innestate, e richiedono semplicemente che le clausole dipendenti siano adiacenti ai costituenti da cui dipendono.

Questo tipo di grammatica è detta context-free poiché una produzione della forma:

$$A \rightarrow x$$

significa che A può essere sostituita da x ovunque essa appaia, senza considerare il contesto (dei simboli vicini ad A).

La scelta di una grammatica context-free è stata dettata da tre importanti caratteristiche:

- Fornisce una precisa definizione matematica che scarta alcuni tipi di linguaggio;
- La definizione formale implica che le CFG sono computazionalmente trattabili, ovvero è possibile scrivere un programma che determina se le sentenze sono grammaticalmente corrette o meno;
- Permette una notazione per una semplice visualizzazione della struttura delle sentenze tramite un albero.

L'utilizzo del linguaggio Prolog permette di esprimere la grammatica CFG tramite la notazione DCG (Definite Clause Grammar).

Le Definite Clause Grammars sono un'estensione delle grammatiche libere da contesto e rappresentano descrizioni eseguibili di una grammatica.

La semantica delle DCG può essere data traducendole nei corrispondenti programmi Prolog, che sono analizzatori sintattici per le grammatiche stesse.

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



Nel formalismo delle DCG una grammatica è rappresentata come un insieme di fatti Prolog dove:

- ogni produzione è un termine Prolog con funtore --> (operatore binario infisso);
- i simboli non terminali sono rappresentati come atomi;
- i simboli terminali sono rappresentati tra parentesi quadre;
- [ ] indica la stringa vuota.

Un esempio di grammatica che definisce un frammento di linguaggio naturale (lo scopo è il non terminale frase) è il seguente:

```
frase --> fraseNominale, fraseVerbale.  
fraseNominale --> articolo, nome, fraseRelativa.  
fraseVerbale --> verbo, fraseNominale.  
fraseRelativa --> [che], fraseVerbale.  
fraseRelativa --> [].  
articolo --> [il].  
articolo --> [un].  
nome --> [cane].  
nome --> [gatto].  
verbo --> [ama].  
verbo --> [mangia].
```

Una grammatica espressa mediante DCG può essere poi facilmente tradotta in un programma Prolog che definisce l'analizzatore della grammatica, tale programma contiene una clausola per ogni produzione:

```
frase --> fraseNominale, fraseVerbale.  
frase(ListIn,ListOut):- fraseNominale(ListIn, ListTemp),  
fraseVerbale(ListTemp,ListOut).
```

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



articolo --> [il].

articolo([il|List],List).

Per descrivere una grammatica libera dal contesto, occorre poi definire il vocabolario contenente tutte le parole che possono essere utilizzate per costruire le frasi che la grammatica può riconoscere. La tecnica utilizzata per realizzare il vocabolario è quella *non normalizzata*, che riporta tutti i lemmi nelle varie declinazioni in riferimento al genere e al numero (esempio *agg(bello).*, *agg(belli).*, ecc.); in questo modo le informazioni sono contenute in un unico predicato.

Le parole, come detto, vengono classificate in base alle parti del discorso: nomi, articoli, aggettivi, pronomi, verbi, avverbi, congiunzioni, preposizioni, interiezioni. Pertanto, secondo un primo approccio, ogni parola può essere considerata un simbolo terminale, che opportune regole, descritte secondo le notazioni dei linguaggi formali, associano alla corrispondente parte del discorso; se un elemento della frase può comparire in forme alternative, è necessario scrivere una opportuna regola per ciascuna di esse.

L'analizzatore consente di compiere una analisi sintattica della frase, non solo verificando se la frase è costruita in modo coerente con le regole descritte nel programma, ma restituendo anche un'informazione relativa alla struttura della frase analizzata, rappresentata dal *parse tree* (o albero sintattico).

#### 1.2.4 TRIPLE EXTRACTOR

A questo componente è demandato la estrazione di triple analizzando i sintagmi presenti nell'albero sintattico.

L'idea alla base dell'approccio è quella di estrarre dalla query in linguaggio naturale, i principali sintagmi (nominali, verbali, preposizionali, etc.) riconosciuti utilizzando il *Parse Tree*. Ottenuto uno scheletro della query formato da tutti i sintagmi riconosciuti in essa, se ne estraggono tutti i sintagmi nominali e si ricercano le relazioni esistenti tra ogni coppia di essi. Queste relazioni possono essere sintagmi verbali, sintagmi proposizionali, congiunzioni o anche altri sintagmi nominali.

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



La coppia di sintagmi nominali costituisce il *soggetto* e l'*oggetto* di una tripla nella forma di *<soggetto, predicato, oggetto>* mentre la relazione esistente tra la coppia ne rappresenta il *predicato*.

Allo stesso tempo nelle ontologie, i fatti che modellano un particolare dominio, sono descritti anche essi tramite delle triple nella forma di *<soggetto, predicato, oggetto>*. Il *soggetto* e l'*oggetto* possono essere classi, istanze o valori letterali, le quali sono espresse tramite nomi, sintagmi nominali o frasi. Il *predicato* esprime le relazioni (le proprietà) esistenti tra le classi, le istanze o i valori letterali, e possono essere espressi tramite un verbo o un sintagma verbale. Di conseguenza ogni coppia di sintagmi nominali e la relazione intercorrente tra di loro (che come detto può essere un sintagma verbale, un sintagma preposizionale o una congiunzione) individuate nelle query possono essere mappate sulle triple di una ontologia.

Queste triple sono dunque individuate tramite una ricerca di veri e propri pattern all'interno della query, in questo caso del tipo SN-SV-SN.

Il pattern esposto precedentemente rappresenta senza dubbio il caso più semplice, ma anche il più ricorrente.

Attraverso delle euristiche è però possibile riconoscere all'interno della query utente degli ulteriori pattern, i quali vengono descritti nel paragrafo successivo.

Naturalmente le triple estratte sono in una forma grezza, in quanto i sintagmi nominali e le relazioni individuate contengono parole di ogni genere (articoli, preposizioni, nomi, verbi, etc.), per questo motivo, da ogni sintagma vengono filtrate quelle parole considerate non necessarie, quali articoli, preposizioni, ecc., e mantenute solo i nomi e i verbi.

I pattern individuati devono inoltre sovrapporsi, sul primo o sull'ultimo sintagma, in modo da permettere l'unione delle triple per poter formare il grafo query che serve per la fase successiva di matching.

#### 1.2.4.1 Heuristic Pattern Detection

Come descritto nel paragrafo precedente, il pattern più frequente all'interno della query è del tipo SN-SV-SN, in quanto rappresenta la forma più frequente con cui si presenta una espressione in linguaggio naturale. Ovviamente data la complessità di una lingua come

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

quella italiana, i pattern individuabili all'interno di una frase sono molti di più e delle forme più varie.

Di seguito se ne descrivono alcuni che vengono riconosciuti all'interno del tool.

Nel pattern SN-SV-SN il ruolo di predicato è svolto da un verbo o da un sintagma verbale, ma all'interno di una ontologia tale ruolo può essere espresso anche per mezzo di preposizioni (sintagmi preposizionali) o di nomi (sintagmi nominali e aggettivali). Dunque è possibile mappare su delle triple anche dei pattern del tipo SN-SP-SN, SN-SN-SN e SN-SA-SN, dove il primo sintagma svolge il ruolo di soggetto, il secondo svolge il ruolo di predicato mentre il terzo il ruolo di oggetto.

Un altro pattern ricorrente è formato dal solo sintagma nominale SN formato da un nome e da un aggettivo qualificativo. Tale aggettivo può avere la valenza o di attributo o di specificazione.

Di conseguenza un simile pattern è mappabile su di una tripla in due differenti modi:

- Il termine ha una chiara valenza di specificazione di un concetto, di conseguenza si crea una tripla in cui il nome è la superclasse e rappresenta l'oggetto, il predicato è del tipo *subClassOf*, mentre il nome unito all'aggettivo è la sottoclasse e rappresenta il *soggetto*;
- Il termine ha valenza di attributo quindi si crea una tripla in cui il nome è il soggetto e l'aggettivo è l'oggetto, mentre il predicato viene lasciato vuoto e sarà compito del *Semantic Mapper* ricercare un predicato opportuno.

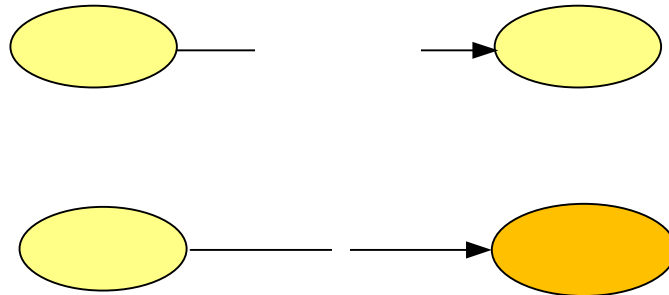
Esempio:

Linguaggio naturale:

SN
La birra rossa

Traduzione ontologica:

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



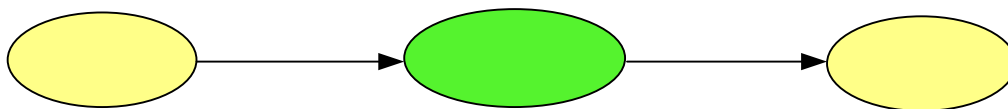
Un altro pattern è del tipo SN-SV-SN-SN dove il sintagma verbale è formato SV da un verbo ausiliare (nella lingua italiana il verbo essere o avere). In tal caso il SV viene concatenato con il successivo SN formando una unica entità SN-(SV SN)-SN e quindi verrà mappata su di una singola tripla, dove il ruolo di predicato è svolto dall'unione del sintagma verbale e nominale. Il tutto è basato sulla assunzione che un predicato in un'ontologia difficilmente è espresso mediante l'utilizzo del solo verbo ausiliare, ma molto probabilmente esso è concatenato con un nome che meglio ne definisce il significato.

Esempio:

Linguaggio naturale:

SN	SV , SN	SN
La birra	ha , come ingrediente	<b>birra</b> il malto

Traduzione ontologica:



Un'altra tipologia di pattern è del tipo SN-SV-SN dove il sintagma verbale è formato SV dal verbo essere. Questo pattern ha la chiara valenza di esistenza di una relazione di sottoclasse tra il primo ed il secondo SN (e quindi permette anche una forma di disambiguazione del primo SN), pertanto si crea una tripla in cui il primo SN rappresenta

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

la sottoclasse, il predicato è del tipo subclassOf, mentre il secondo SN rappresenta la superclasse.

Esempio:

Linguaggio naturale:

SN	SV , SN
La birra	è una bevanda

Traduzione ontologica:



Il primo sintagma nominale può anche essere formato da un nome proprio anziché da un nome semplice, di conseguenza tale sintagma ha la valenza di istanza del concetto espresso dal secondo sintagma.

Esempio:

Linguaggio naturale:

SN	SV , SN
La Lager	è una birra

Traduzione ontologica:



Un altro pattern è del tipo SN-SP-SN dove il sintagma preposizionale è formato dalla preposizione *di*. In analisi logica tale sintagma ha il significato di complemento di specificazione ed ha la funzione di precisare un termine che altrimenti resterebbe generico. Tale funzione non è molto lontana da quella dell'attributo, che aggiunge una determinazione a un nome: in molti casi, infatti il complemento di specificazione può essere sostituito da un aggettivo corrispondente.

Dopo queste considerazioni è evidente che un simile pattern è mappabile su di una tripla in due differenti modi:

- Il termine ha una chiara valenza di specificazione di un concetto, di conseguenza si crea una tripla in cui il primo SN è la superclasse e rappresenta l'oggetto, il predicato è del tipo subclassOf, mentre l'unione di SN e SP è la sottoclasse e rappresenta il *soggetto*;
- Il termine ha valenza di attributo quindi si crea una tripla in cui il SP è il predicato e il SN è il soggetto, mentre il secondo SN è l'oggetto.

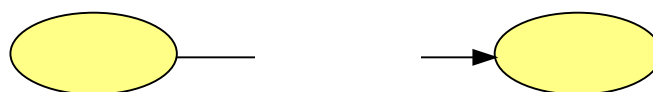
Di conseguenza vengono generate due triple.

Esempio:

Linguaggio naturale:

SN	SP	
		SN
La birra	di	colore rosso

Traduzione ontologica:





### 1.2.5 SEMANTIC MAPPER

A questo componente del sistema è affidato il compito di mappare le triple estratte dalla query in linguaggio naturale, ottenute nella fase precedente, sulle triple definite nelle ontologie. Il componente utilizza algoritmi basati sulla similarità tra stringhe, risorse lessicali, dizionari (Wordnet) e risorse linguistiche pensate ad hoc per un particolare dominio.

Le ontologie con le quali fare il confronto sono tutte quelle che hanno una certa attinenza con la query formulata dall'utente, cioè quelle ritornate dall'*ontology Manager*. Questa operazione si è resa necessaria per garantire una migliore scalabilità del sistema, altrimenti dovrebbero essere sottoposte all'operazione di mapping anche ontologie che non hanno nulla a che vedere con il vocabolario utilizzato dall'utente per la query.

Le triple ricavate nella fase precedente vengono unite dal Semantic Mapper per formare un grafo che verrà poi mappato sul grafo dell'ontologia.

Per intraprendere l'operazione di mapping, i singoli componenti delle triple (principalmente nomi e verbi) vengono "matchati" con le entità (classi, relazioni e istanze) dell'ontologia in esame.

Per far questo vengono impiegate diversi metodi di matching che possono essere classificati in due tipi:

- *matching semantico*: utilizza dizionari come WordNet o altre Ontologie (Upper Ontologies) per trovare sinonimi e collegamenti semantici, tra le parole della query e le entità dell'ontologia;
- *matching sintattico*: utilizza metriche di similarità tra stringhe o altre euristiche (ad esempio dizionari che contengono abbreviazioni di particolari espressioni) per trovare una similarità non esatta tra le parole della query e le entità dell'ontologia.

Il mapping ottenuto tra i componenti della query e le entità dell'ontologia non deve necessariamente rispettare il pattern della tripla estratta dalla query utente. Ad esempio un

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

sintagma nominale che era stato associato al soggetto della tripla potrebbe essere mappato su di un predicato (relazione) dell'ontologia.

Prima di introdurre la tecnica utilizzata per l'estrazione dei sottografi, bisogna specificare la struttura dati interna che è stata utilizzata per effettuare l'operazione di matching.

La struttura utilizzata è un grafo diretto aciclico, in cui i nodi possono corrispondere sia a concetti (classi in Owl) sia a relazioni (proprietà in Owl), mentre gli archi, che collegano tra di loro i nodi, di conseguenza, collegano una classe ad un'altra classe o una classe ad una relazione (o viceversa). Utilizzando una tale rappresentazione, si effettuerà una operazione di matching sintattico e/o semantico solo sui nodi, in quanto gli archi avranno una label associata che è assolutamente fissa e priva di possibili ambiguità. In altre parole, gli archi possono assumere come label o gli assiomi Owl (*subClassOf*, *subPropertyOf*, *unionOf*, ecc.) o, se si riferiscono a relazioni, le definizioni di dominio e codominio (*domain* e *range*), di conseguenza un match tra archi può essere solo di uguaglianza stretta (0 o 1). D'altro canto, gli archi sono tenuti in conto nel momento in cui si effettua un match di tipo strutturale.

### 1.2.5.1 Subgraph Extraction

La tecnica implementata si basa sulla ricerca ed estrazione, dall'Ontology Graph, dei sottografi che sono in comune con il Query Graph.

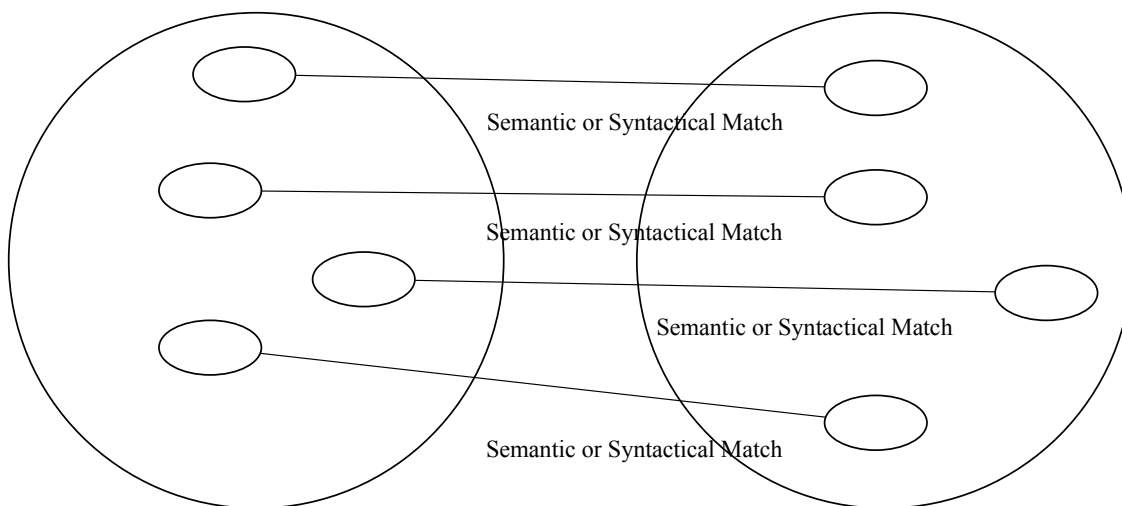
Questa tecnica è suggerita dal fatto che alcune query hanno come obiettivo quello di estrarre informazioni relative alla struttura dei dati, specificando così dei vincoli che qualificano meglio i risultati ottenuti. Lo scopo di queste query è quindi estrarre le relazioni tra le entità del dominio, vale a dire estrarre sottografi o quanto meno dei path dal grafo dei dati (in questo caso l'ontologia).

La tecnica sviluppata è stata formulata per tenere in conto sia i risultati della precedente fase di matching sintattico-semantico, sia tenendo in conto la possibilità di avere delle condizioni di matching strutturale rilassate.

In ingresso all'algoritmo si hanno tutti i nodi che hanno una similarità sintattica o semantica dei due grafi, questi nodi non sono però tra di loro correlati all'interno del singolo grafo. In altre parole, si ha a disposizione una nuvola di concetti e relazioni del grafo query ed una nuvola di concetti e relazioni del grafo ontologico, i cui nodi interni non sono tra di loro

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

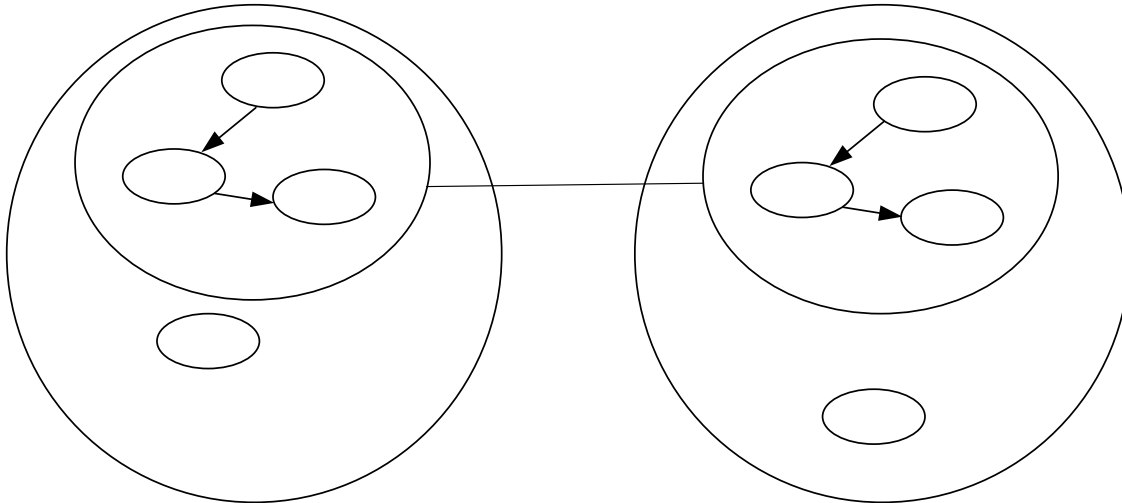
collegati (o meglio i collegamenti non sono stati sottoposti all'operazione di match, quindi non hanno delle corrispondenze).



Si parte dunque da un nodo del grafo query e si cerca, navigandone gli archi, di trovare delle corrispondenze anche nel grafo ontologico. Più grande sarà il sottografo comune trovato, maggiore sarà il risultato di matching strutturale.

Naturalmente possono essere trovati più sottografi in comune.

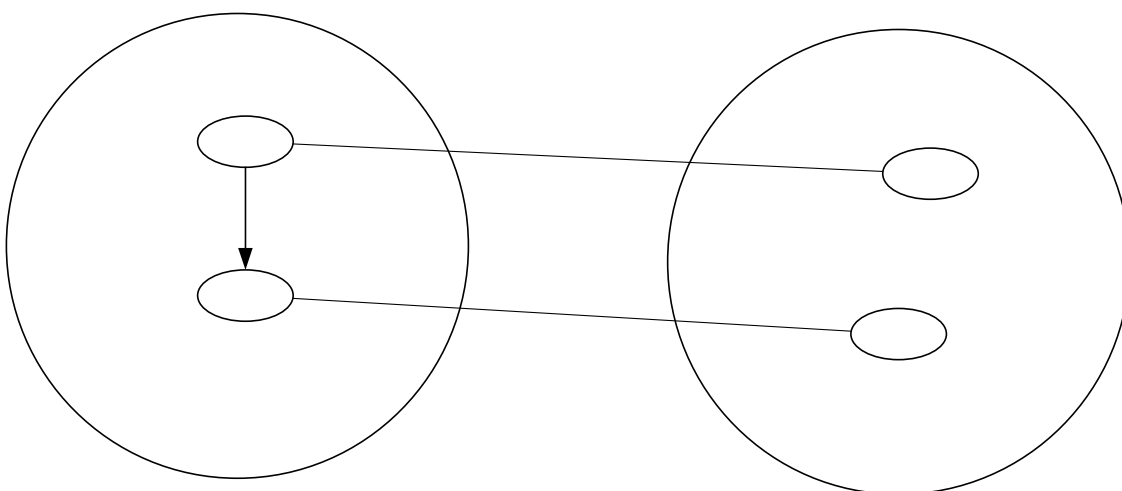
PROGETTO LC3	Revisione n*	0	Del	-----
--------------	--------------	---	-----	-------



Il precedente è il più semplice tipo di matching strutturale, in quanto i due sottografi coincidono (sono isomorfi), è possibile invece trovare ulteriori matching strutturali nel momento in cui gli archi tra i nodi non corrispondano.

Consideriamo il caso in cui si abbia trovato due nodi coincidenti per ogni grafo e si voglia trovare anche una corrispondenza strutturale (quindi si cerca un matching tra archi).

Structural Ma





Nella fase precedente abbiamo trovato un match tra due nodi dei due diversi grafi, ora vogliamo trovare un possibile matching strutturale analizzando gli archi tra i nodi del grafo query e cercandone una corrispondenza in quello del grafo ontologico. In questo caso si nota che nel grafo ontologico i due nodi non sono collegati esplicitamente, quindi una tecnica di sottografo isomorfismo non è applicabile.

E' possibile a questo punto applicare delle tecniche di inferenza utilizzando l'ontologia, in modo da poter estrarre un sottografo dall'Ontology Graph.

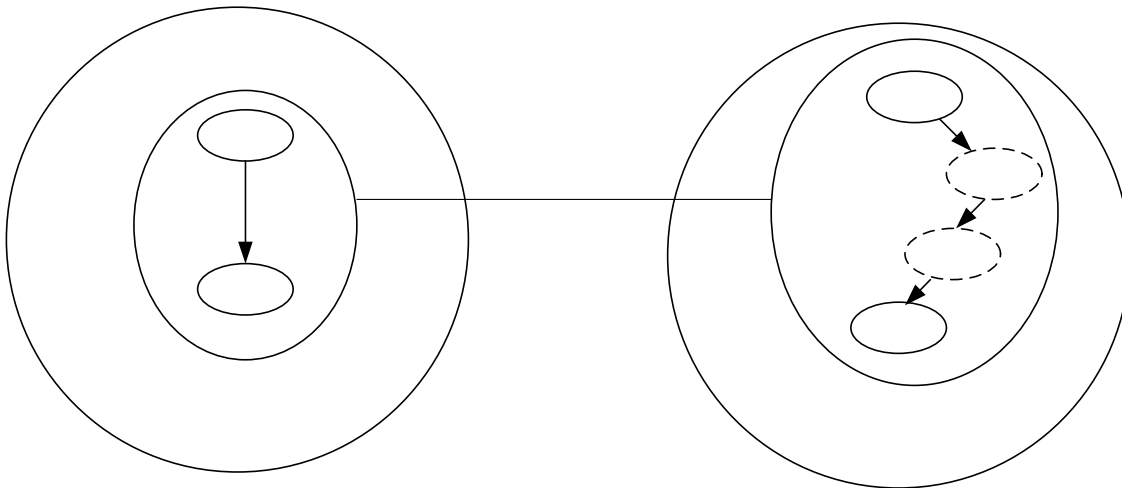
Nella tecnica sviluppata attualmente, si analizzano le gerarchie di tipo generalizzazione/specializzazione, cioè quelle che implicano l'attraversamento di relazioni del tipo *SubClassOf*, per poter verificare se una determinata classe è riferibile ad una proprietà che il Query e l'Ontology Graph hanno in comune.

Supponiamo che il primo nodo nel Query Graph sia una classe e sia legato ad un secondo nodo che è una proprietà: in questo caso si individua nell'Ontology Graph il nodo classe a cui è riferita la proprietà, cioè il suo dominio, e si cerca tra le sue sottoclassi (dirette o non) se vi sia il nodo che ha matchato con quello del Query Graph. Se la ricerca ha esito positivo allora può essere estratto dall'ontologia il sottografo formato dalla proprietà e dalla sottoclasse. Tale operazione è lecita in quanto in un'ontologia le sottoclassi ereditano tutte le proprietà della superclasse.

Questa tecnica è utilizzabile anche nel caso simmetrico, cioè il primo nodo nel Query Graph è una proprietà ed è legato ad un secondo nodo che è una classe: in questo caso nell'Ontology Graph si individua il nodo classe a cui è riferita la proprietà, cioè il suo range, e si procede similmente al caso precedente.

Nel caso generale bisognerebbe individuare un possibile path, il minimo per esattezza, tra i due nodi nell'Ontology Graph e quindi la relativa lunghezza. Nel caso in cui si individuino più path con la stessa lunghezza sarebbe demandato all'utente la selezione di quello più rispondente ai suoi scopi.

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



Nell'esempio si nota che ad un arco del primo grafo corrispondano nel secondo grafo due nodi e tre archi.

Queste tipologie di matching strutturale "rilassate" hanno però l'inconveniente di richiedere molto più tempo computazionale rispetto a quelle esatte, in quanto bisogna accedere all'ontologia e ricercare un cammino, il più piccolo possibile tra i due nodi.

In base a questa considerazione, è dunque possibile lasciare all'utente la possibilità di poter scegliere quale tecnica di matching utilizzare: solo sui singoli nodi (e tra queste scegliere se sintattica o semantica), strutturale stretta o rilassata.

Una volta individuati i sottografi comuni questi vengono ritornati all'utente e resi disponibili per la successiva fase di *Visual Query* in cui l'utente può raffinare la Query aggiungendovi ulteriori concetti e/o relazioni.

### 1.2.6 TARGET AND MODIFIER EXTRACTOR

Per poter tradurre una query in linguaggio naturale in una query SPARQL, bisogna trovare i *Target*, cioè le parole che corrispondono alle variabili dopo la clausola SELECT nella query SPARQL. Il procedimento è effettuato come segue: prima si cercano gli aggettivi o i pronomi interrogativi come "quale", "chi", "come" e così via; dopo si scelgono i nomi che seguono tali locuzioni interrogative come target.

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

In SPARQL è possibile utilizzare il modificatore *Filter* per permettere agli utenti di specificare delle restrizioni sulle variabili e/o sulle relazioni tra di esse. Altri modificatori forniscono all'utente la possibilità di modificare i risultati ritornati da una query (tramite l'uso di *Order by*, *Limit* e *Offset*).

Nel linguaggio naturale questi modificatori sono espressi tramite certi tipi di parole o espressioni, che possono essere identificate nella query. Si riportano di seguito le parole o le espressioni identificate nella query:

1. Negazioni: tramite le parole “non” e “no”;
2. Comparativi e superlativi: tramite gli avverbi “più” e “meno” seguiti da aggettivi qualificativi oppure riconoscendo il grado degli aggettivi tramite i risultati del Parse Tree;
3. Congiunzioni: tramite le parole “e” e “o”.

Una volta riconosciuti tali espressioni, esse vengono memorizzate e passate alla fase di SPARQL Generator per essere tradotte opportunamente in linguaggio SPARQL.

### 1.2.7 VISUAL QUERY BUILDER

In questa fase vengono presentati all'utente i risultati della fase di matching, cioè un sottografo che ha matchato con la query in linguaggio naturale. L'utente può direttamente sottoporre la query allo *SPARQL Engine* o al *Graph Matching Engine* oppure validarla andando ad aggiungere/eliminare classi e/o relazioni. La query è visualizzata tramite una notazione a grafo di cui se ne riporta la sintassi utilizzata.

#### 1.2.7.1 Notazione Grafica

Poiché le query in SPARQL utilizzano la struttura basata su triple di RDF, una rappresentazione che utilizzi una sequenza di nodi e link può essere utilizzata per rappresentare molte tipologie delle query SPARQL. I nodi rappresentano nel nostro caso Concetti (classi Owl) mentre gli archi collegano tra loro concetti o un concetto ed una proprietà (object e datatype property in owl).

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

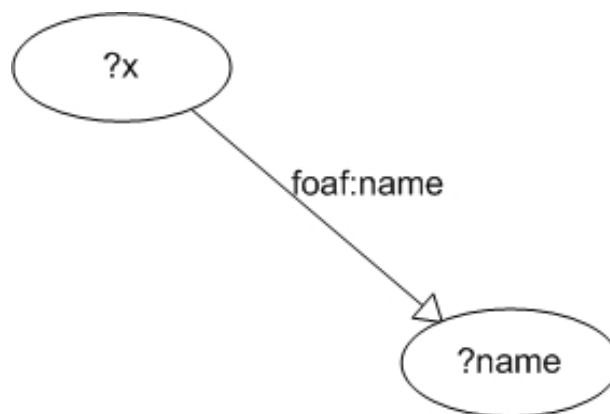
Una volta che la query è stata formulata graficamente, questa viene parserizzata e tradotta in automatico in linguaggio SPARQL, secondo le regole definite di seguito.

### 1.2.7.2 Triple Pattern Base

I nodi corrispondono ad URI, valori letterali o variabili (bound o unbound, a seconda che appaiono o meno nella clausola SELECT). I nodi sono rappresentati graficamente come rettangoli ed hanno un colore diverso a seconda del tipo (URI, letterali o variabili). Inoltre hanno associata una label che ne indica il valore. I nodi corrispondono a classi owl.

Gli archi sono rappresentati tramite linee e sono utilizzati per definire le relazioni tra classi (Object Property) o gli attributi di una classe (Datatype Property). Ad essi sono associate una label che ne indica il nome. L'orientamento dell'arco indica quale nodo sia il dominio di una proprietà, cioè la classe ai cui individui appartenenti si può applicare la proprietà, e quale sia il range (o codominio) di una proprietà, cioè la classe i cui individui appartenenti possono essere valori della proprietà.

Esempio:



Sintassi Sparql corrispondente:

```

SELECT ?x ?name
WHERE
{
    ?x foaf:name ?name
  
```

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

}

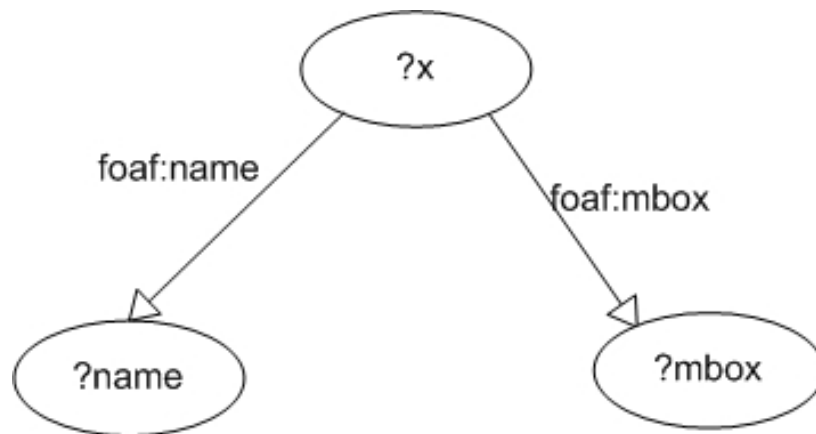
se ?x è selezionata come variabile un-bound allora non compare nella clausola SELECT:

```
SELECT ?name
WHERE
{
    ?x foaf:name ?name
}
```

#### 1.2.7.4 Triple Patterns Multipli

In una query possono essere specificati più triple pattern attraverso l'aggiunta di più nodi ed archi. Se ci sono più variabili o letterali condivisi nei pattern, allora questi sono rappresentati utilizzando un unico nodo con archi multipli.

Esempio:



Sintassi SPARQL corrispondente:

```
SELECT ?x ?name ?mbox
WHERE
{
    ?x foaf:name ?name.
```

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



```
?x foaf:mbox ?mbox.  
}
```

### 1.2.7.5 Graph Pattern

In SPARQL, un graph pattern consiste in una o più triple pattern che sono matchati sull'intero grafo Rdf/Owl. Tali graph pattern influenzano il binding delle variabili dato che ogni variabile ha uno scope locale rispetto al graph pattern nelle quali sono contenute. Ciò significa che una stessa variabile può essere matchata con differenti valori nei differenti graph pattern. L'utilizzo dei graph pattern permette di matchare le triple contenute in esso con l'intero grafo owl e tale match non è influenzato da un precedente graph pattern. Graficamente quando un nodo compare in graph pattern multipli i nodi sono duplicati, anche se internamente sono trattati come se fossero uno solo (rispetto ad ogni graph pattern).

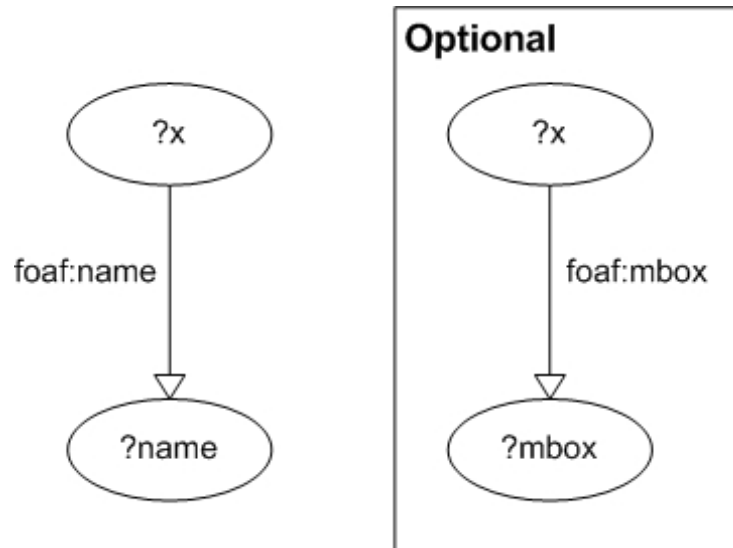
Nel seguito vedremo due tipologie di graph pattern.

### 1.2.7.6 Optional Graph Pattern

Un graph pattern opzionale è semplicemente visualizzato tramite un raggruppamento sottoforma di rettangolo con l'etichetta OPTIONAL. Nell'esempio seguente il grafo di destra è obbligatorio, nel senso che è necessario che si trovi nel grafo owl, il secondo grafo a destra è opzionale, nel senso che non è necessario che si trovi nell'owl.

Esempio:

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



Sintassi SPARQL corrispondente:

```

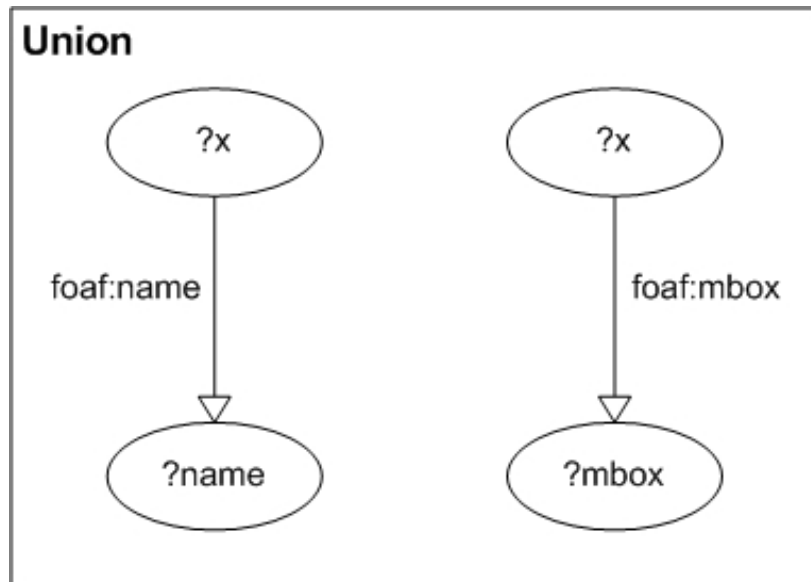
SELECT ?x ?name ?mbox
WHERE
{
    ?x foaf:name ?name.
    OPTIONAL {?x foaf:mbox ?mbox.}
}
    
```

### 1.2.7.7 Union Graph Pattern

La rappresentazione visuale di un union graph pattern (cioè di un graph pattern dove uno dei due grafi può essere considerato come parte della soluzione della query) è ottenuta unendo i due graph pattern in un unico raggruppamento sottoforma di rettangolo con l'etichetta UNION.

Esempio:

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



Sintassi SPARQL corrispondente:

```

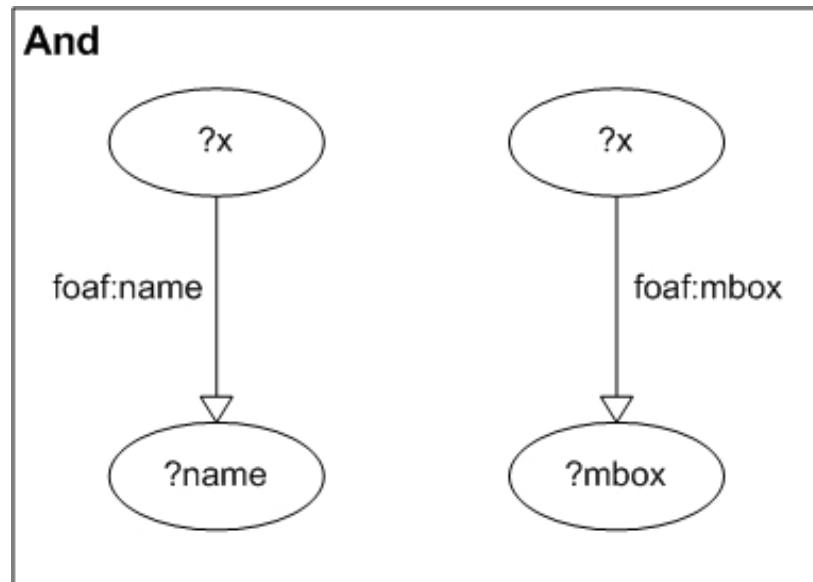
SELECT ?x ?name ?mbox
WHERE
{
    {?x foaf:name ?name.}
    UNION
    {?x foaf:mbox ?mbox.}
}
    
```

### 1.2.7.8 And Graph Pattern

La rappresentazione visuale di una and graph pattern (cioè di un graph pattern dove entrambi i due grafi devono essere parte della soluzione della query) è ottenuta unendo i due graph pattern in un unico raggruppamento sottoforma di rettangolo con l'etichetta AND.

Esempio:

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



Sintassi SPARQL corrispondente:

```

SELECT ?x ?name ?mbox
WHERE
{
    ?x foaf:name ?name.
    ?x foaf:mbox ?mbox.
}
    
```

Come si nota dalla sintassi SPARQL, l'and di due grafi corrisponde esattamente al graph pattern multiplo, in quanto ogni tripla in SPARQL è collegata tramite una and logica. Con tale costrutto è però possibile associare due variabili distinte ai due grafi.

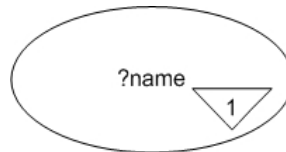
### 1.2.7.9 Ordinamento dei risultati

I risultati della query SPARQL possono essere ordinati in base a qualsiasi variabile. Per rappresentare ciò visualmente, viene utilizzato un indicatore a forma di freccia, indicante il tipo di ordinamento (ascendente o discendente), con un numero all'interno, indicante la posizione della variabile nella clausola ORDER BY.

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

Se l'indicatore non è presente, la variabile non è usata per ordinare i risultati della query.

Esempio:



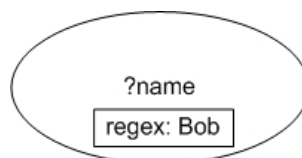
Sintassi SPARQL corrispondente:

```
SELECT ?name
WHERE
{
    ...
}
ORDER BY DESC (?name)
```

### 1.2.7.10 Filtraggio delle variabili

In SPARQL il filtraggio è utilizzato per restringere l'insieme dei risultati ritornato da una query, utilizzando delle espressioni numeriche o letterali. La rappresentazione visuale prevede l'utilizzo di un box all'interno del nodo o del link che partecipa all'operazione di filtraggio.

Esempio:



Sintassi SPARQL corrispondente:

```
SELECT ?name
```

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



```
WHERE
```

```
{  
  
    ...  
  
    FILTER (REGEX(?name, "Bob"))  
  
}
```

### 1.2.7.11 Distinct, Limit e Offset

SPARQL fornisce anche funzionalità per ritornare valori distinti (Distinct), limitare il numero di risultati ad un numero prefissato (Limit) e per specificare l'indice iniziale da cui prelevare i risultati (Offset). Queste opzioni sono globali alla query e sono settate con un apposito menù in quanto non si riferiscono al particolare nodo o link.

### 1.2.8 ONTOLOGY QUERYING

Una volta formulata la query questa può essere sottoposta ad un motore di query semantico, a tale scopo sono stati scelti il motore di query ARQ di Jena ed il motore di query del reasoner Pellet, in entrambi i casi il grafo rappresentante la query deve essere parserizzato nel linguaggio SPARQL.

Jena mette a disposizione diversi tipi di reasoner che si differiscono per livello crescente di complessità e decrescente per quanto riguarda la velocità di esecuzione.

Naturalmente, i primi sono quelli più semplici ed hanno una capacità di deduzione più limitata:

- *Transitive Reasoner*: è capace di individuare e ragionare sulla gerarchia tra classi e proprietà definita attraverso i costrutti *rdfs:subClassOf* e *rdfs:subPropertyOf*;
- *RDFS Reasoner*: aggiunge il supporto per i costrutti *rdfs:range* e *rdfs:domain*;
- *OWL, OWL Mini, OWL Micro Reasoners*: offrono un sottoinsieme delle operazioni di OWL.

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



Jena dispone dunque di tre reasoner interni per OWL di complessità crescente. Purtroppo anche il più complesso dei tre riesce a malapena a trattare un sottoinsieme di OWL-DL molto vicino ad OWL-Lite.

Per ovviare a questa mancanza Jena rende possibile la connessione con reasoner esterni conformi allo standard DIG oppure una connessione diretta a reasoner che implementino l'interfaccia *Reasoner*.

La scelta è stata di utilizzare, all'interno del tool sviluppato, Pellet come reasoner esterno, collegandolo attraverso le Jena API fornite dal reasoner stesso. Queste librerie forniscono una specializzazione dell'interfaccia *Reasoner* di Jena apposita per tale componente.

Utilizzando l'interfaccia grafica del tool sviluppato, è possibile scegliere quale motore di query utilizzare e nel caso di Jena, è possibile specificare quale tipo di reasoner utilizzare.

### 1.2.9 GRAPH MATCHING QUERYING

La query formulata può essere sottoposta anche ad un motore di matching semantico appositamente sviluppato, il quale cerca le corrispondenze tra il grafo rappresentante la query (di seguito chiamato Query Graph) e i grafi delle annotazioni presenti nel repository delle annotazioni (di seguito denominati Annotation Graph).

Il matching tra Query Graph e Annotation Graph è effettuato tramite tecniche di ontology matching sia di tipo semantico che strutturale.


Data la complessità computazionale di una tale operazione di matching, visto il numero non predicibile di annotazioni presenti, queste sono anch'esse indicizzate.

Il Query Graph ha settato un nodo *entry*, cioè il nodo da cui parte il processo di matching; tale scelta è ottenuta grazie al rilevamento dei *Target* durante la fase di *Target e Modifier Extraction*, gli Annotation Graph sono invece indicizzati su tutti i concetti e le relazioni che sono state utilizzate durante il processo di annotazione e quindi esse sono tutte *entry*.

Il processo di matching è dunque preceduto dalla ricerca della *entry*, fornita nel Query Graph, all'interno del database delle annotazioni, a questa *entry* sono inoltre associati tutti i suoi sinonimi.

Utilizzando questo approccio si garantisce la scalabilità dell'architettura in quanto la fase di indexing è precedente alla operazione di query.

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

	PARTNER: SECONDA UNIVERSITÀ DI NAPOLI			
	RESPONSABILE PROF. BENIAMINO DI MARTINO			
Technical Report: 2.4.9				
LC3 – Laboratorio pubblico-privato di ricerca sul tema della Comunicazione delle Conoscenze Culturali			PAG 33 DI 37	

Una volta pre-selezionate le annotazioni è possibile effettuare l'operazione di matching, questa volta su di un numero limitato di annotazioni; possono presentarsi vari casi a seconda della complessità strutturale della annotazione e della query:

- nel caso in cui l'annotazione faccia riferimento solo a concetti singoli di una o più ontologie, viene applicato solo un matching di tipo semantico;
- nel caso in cui l'annotazione sia riferita a più concetti legati tra di loro da proprietà (vi è quindi la definizione di un grafo di annotazione) il matching applicato è anche di tipo strutturale, nel senso che un documento è maggiormente rilevante per la query formulata, se l'annotazione che lo descrive ha una struttura "compatibile" al grafo query.

Di seguito vengono definite le tecniche semantiche e strutturali utilizzate nelle operazioni di matching.

### 1.2.9.1 Matching sintattico e semantico

In primis viene definita la differenza chiave tra syntactic e semantic matching. Nel syntactic matching, il match è costruito basandosi solo sull'etichetta associata ai singoli nodi, indipendentemente dalla loro posizione nel grafo (nel nostro caso i nodi devono necessariamente appartenere ad un'ontologia).

Nel semantic matching, invece, quando si cerca un match tra due nodi, tale match non dipende solo dal concetto associato a quel nodo (il concetto che è denotato dall'etichetta del nodo), ma anche dalla posizione del nodo nel grafo.

Nella tecnica implementata, il singolo nodo del Query Graph è confrontato con tutti i nodi del grafo annotazione innanzitutto utilizzando delle metriche di similarità sintattica e semantica (una soglia di similarità, compresa tra 0 ed 1, è definita a priori dall'utente o preimpostata).

Le label dei nodi dei due grafi sono confrontate seguendo questi passi:

1. Verifica di uguaglianza esatta;
2. Utilizzo di Wordnet per verificare se sono sinonimi;

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

3. Vengono sottoposte a stemming e confrontate tramite metriche di similarità tra stringhe (Edit Distance, Euclidean Distance, Jaro-Winkler Distance, ecc.).

Nel caso in cui il risultato ottenuto non superi la soglia di similarità, si utilizza l'ontologia a cui si riferisce il nodo della annotazione in esame per cercare una relazione semantica tra i due nodi.

L'algoritmo prevede che venga prelevata l'ontologia a cui si riferisce il nodo dell'Annotation Graph e si cerca all'interno della stessa ontologia, se è presente il nodo che fa parte del grafo query.

Se il nodo è presente vi sono due alternative possibili:

1. si cerca un cammino (il più breve) che colleghi questo nodo al nodo dell'annotazione, la distanza tra i nodi è calcolata in base al percorso (ai nodi attraversati è associato un peso);
2. si verifica se esiste una relazione gerarchica tra i due nodi, siano essi concetti o proprietà.

Nel caso in cui nessun matching è stato trovato o la soglia di similarità non è stata raggiunta, allora i due nodi non sono simili e si passa al nodo successivo presente nell'annotazione.

Nel caso siano stati trovati più matching per un nodo, si ritorna quello a similarità più elevata.

### 1.2.9.2 Matching strutturale

La tecnica definita per effettuare il matching strutturale, si basa sulla ricerca del massimo comune sottografo. Questa misura di similarità tra grafi confronta due grafi  $G_A$  e  $G_B$  generandone un terzo, il quale è il massimo comune sottografo, vale a dire il grafo più grande contenuto in entrambi i grafi  $G_A$  e  $G_B$ .

A tale scopo viene utilizzato l'algoritmo di McGregor.

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

Data la complessità computazionale di questo algoritmo (i problemi di ricerca del M.C.S. sono noti per essere NP-Completi), il quale dovrebbe essere applicato a partire da tutti i nodi presenti nell'Annotation Graph sotto esame, se ne è definita una versione rilassata; per ottenere ciò si è pensato di riutilizzare l'introduzione del concetto di *entry* di un grafo (definito nel [Paragrafo precedente](#)) in modo che il processo di matching parta esclusivamente da questo nodo *entry*.

In altre parole l'algoritmo verifica se esiste un sottografo comune tra i due grafi, a partire esclusivamente dei due nodi *entry*; nessun altro nodo dei due grafi è scelto come possibile nodi di partenza attraverso il quale effettuare la ricerca del massimo sottografo comune.

Questa restrizione porta ad una diminuzione della complessità pari ad  $n$  dove  $n$  è il numero di nodi del Graph Query.

Infine è stata definita una metrica di similarità tra grafi adattando la metrica di distanza MCS:

$$d_{MCS}(G_1, G_2) = 1 - \frac{|mcs(G_1, G_2)|}{\max(|G_1|, |G_2|)}$$

Poiché si vuole mappare l'intero Query Graph su di un Annotation Graph, (abbiamo quindi introdotto un "verso" nell'operazione di match), invece di considerare al denominatore il numero di nodi del più grande tra i due grafi, consideriamo il numero di nodi del Query Graph, cosicché se esso è completamente mappato su di un Annotation Graph il risultato sia 1.

Di conseguenza si ha:

$$\text{Sim}(G_Q, G_A) = \frac{|mcs(G_Q, G_A)|}{|G_Q|}$$

Tale metrica rappresenta la misura con cui si effettua il ranking delle annotazioni.

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



### 1.2.10 DOCUMENT RETRIEVAL AND PRESENTATION

Compito di questa fase è quello di mostrare graficamente agli utenti i risultati ritornati dalla query utente. I risultati sono raggruppati in forma tabellare e rappresentano le istanze definite all'interno dell'ontologia e/o nelle annotazioni che soddisfano i criteri della query.

Un utente naturalmente si aspetta di trovare dei documenti associati a queste istanze, di conseguenza è prevista una funzione che ritorna i documenti che sono correlati (tramite una annotazione semantica precedente) a queste istanze.

All'utente a seguito di un doppio click su di una particolare istanza, vengono ritornati i documenti relativi alla istanza e vengono mostrate le parti del documento che si riferiscono ad essa.

Dopo che l'utente ha selezionato su di una particolare istanza, questi può visualizzare quale siano i documenti, o le parti di esso, che effettivamente sono stati annotati con quelle istanze e/o concetti.

### 1.2.11 FLUSSO DELLE FASI ELABORATIVE

In questo paragrafo si descrivono come si articolano le diverse fasi elaborative, ognuna delle quali è descritta nella definizione data in precedenza dei componenti dell'architettura, per permettere il Semantic Information Retrieval:

1. Analisi grammaticale e sintattica della query in linguaggio naturale;
2. Estrazione delle triple, dei Target e dei modificatori e costruzione del Query Graph;
3. Mapping semantico tra Query Graph ed Ontology Graph;
4. Validazione della query utilizzando una sintassi visuale;
5. Querying Semantico:
  - a. Interrogazione dell'ontologia tramite traduzione in linguaggio SPARQL del Query Graph ed utilizzo di *engine* semantici quali ARQ (Jena) e Pellet;
  - b. Ricerca di annotazioni tramite Graph Matching strutturale;
6. Visualizzazione dei risultati e dei contenuti annotati.

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

Riassumendo, possiamo schematizzare il flusso delle diverse fasi elaborative come segue:

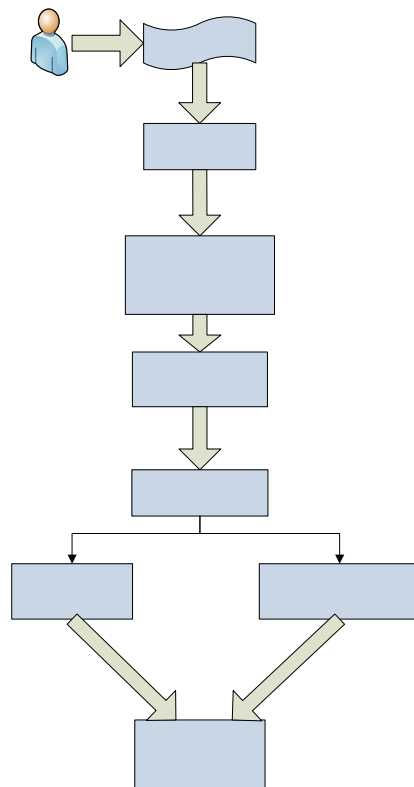


Figura 1.2: Diagramma di flusso delle fasi elaborative del sistema

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

Quer

Syntac  
Parsin

Triple, Ta  
Modifi  
Extract

Semant  
Mappi

Visual Q  
Multi