

Modellazione Concettuale per il Web Semantico

Relazione Progetto

A.A. 2018/2019

Giacomo Costarelli

Giuseppe Gabbia

Dipartimento di Informatica

**Indice**

[**Introduzione e motivazioni 3**](#_Toc6426971)

[**Requisiti 4**](#_Toc6426972)

[**Documentazione 5**](#_Toc6426973)

[**T-box 5**](#_Toc6426974)

[**Object property 7**](#_Toc6426975)

[**Data property 8**](#_Toc6426976)

[**A-box 9**](#_Toc6426977)

[**Visualizzazione dell’ontologia 10**](#_Toc6426978)

[**Query SPARQL 11**](#_Toc6426979)

[**Regole SWRL 14**](#_Toc6426980)

[**Applicazione client 15**](#_Toc6426981)

# 

# **Introduzione e motivazioni**

LinSyn (LINguistic SYNonim) è un’ontologia che nasce come una rappresentazione formale ed esplicita di una concettualizzazione il cui dominio di interesse è la struttura del linguaggio italiano. Insieme a tale rappresentazione vi è il tentativo di orientare l’ontologia stessa ad un task di consultazione con lo scopo di suggerire un insieme di sinonimi per uno specifico vocabolo.

Più nel dettaglio LinSyn, nonostante la sua formulazione teorica, si presta al ragionamento automatico al fine di stabilire quali sono le proprietà peculiari di un particolare vocabolo.

Essendo le ontologie comunemente applicate nel campo dell’intelligenza artificiale e nella rappresentazione e condivisione della conoscenza, LinSyn potrebbe essere utilizzata per una varietà di scopi tra cui il ragionamento deduttivo, la classificazione e diverse tecniche di problem solving in modo da poter facilitare la comunicazione e lo scambio di informazioni fra sistemi diversi.

LinSyn è stata sviluppata attraverso un software open-source scritto in Java e sviluppato dall’università di Stanford di nome Protégé.

L’ontologia è stata codificata attraverso l’utilizzo del linguaggio di markup OWL (Web Ontology Language) che rappresenta l’attuale standard per lo sviluppo di ontologie. La costruzione di LinSyn è stata influenzata da ontologie linguistiche esistenti come GOLD (General Ontology for Linguistic Description) e OnLiT (Ontology for Linguistic Terminology) che descrivono e formalizzano le categorie e le relazioni basilari usate nella descrizione dell’linguaggio umano.

Per lo sviluppo del task individuato, si è anche fatto riferimento ad una banca dati lessicale per l’italiano scritto chiamata CoLFIS, realizzata nel 1995 presso l'istituto di Scienze e Tecnologie della Cognizione di Roma. CoLFIS si basa su un corpus bilanciato di oltre tre milioni di parole, che rispecchia le tendenze di letture degli italiani desunte da dati ISTAT. Inoltre, i dati lessicali estratti dal corpus sono compiutamente lemmatizzati e annotati rispetto alle parti del discorso.

In particolare, CoLFIS fornisce un lessico/dizionario di frequenza sia per i lemmi (lemmario) sia per le forme (formario) i cui dati sono stati integrati in parte nell’ontologia sviluppata.

Per comprendere le differenze alla base dell’ambito, dell’uso e del significato di dati linguistici è necessario uno strumento che fornisca una base terminologica chiarificatrice capace di descrivere una particolare risorsa linguistica. LinSyn nasce come uno strumento per rendere possibile questo modello di rappresentazione dei termini e dei concetti linguistici connessi semanticamente, fornendo una base e presentandosi come uno step da affrontare necessariamente prima di affacciarsi al mondo della semantica.

Il dominio è stato scelto poiché di interesse comune ad entrambi gli studenti che l’hanno progettata ed inoltre, seguendo un percorso di studi interconnesso all’intelligenza artificiale, LinSyn verrà sicuramente utilizzata in progetti futuri come base e modello di dati che potrebbe venire fornito ad un sistema di machine learning per effettuare operazioni come la classificazione automatica o l’analisi di vocaboli e il suggerimento di un loro utilizzo secondo il contesto in cui l’intelligenza è inserita.

Inoltre, la progettazione di LinSyn sembrava, ancor prima di iniziare a progettarla, un task molto complesso ma allo stesso tempo stimolante. Riuscire a completare il lavoro insito nella progettazione e nella effettiva creazione di LinSyn ci ha impegnati ma lasciati soddisfatti del prodotto finito.

La creazione di LinSyn ci ha imposto un approfondimento di ciò che riguarda la morfologia e la fonetica della lingua italiana, il quale è stato effettuato principalmente sull’enciclopedia Treccani e su libri di grammatica italiana. Questo ci ha permesso di affrontare la progettazione dell’ontologia in modo che fosse possibile selezionare e astrarre da un dominio così articolato i moduli e le sezioni più rilevanti poiché la riproduzione completa del dominio di interesse sarebbe andata ben oltre i fini accademici.

L’ontologia, per essere creata correttamente, richiedeva la corretta scelta delle classi da includere in modo da sezionare il dominio senza però tralasciare elementi fondamentali.

Dopo un’approfondita analisi abbiamo deciso di toccare tre macro-aree riguardanti la linguistica italiana ovvero fonologia, morfologia ed unità linguistiche.

# **Requisiti**

L’idea iniziale per la creazione di LinSyn era la suddivisione di quest’ultima in cinque macro classi, *Fonologia*, *Morfologia*, *Sintassi*, *Semantica* e *Unità linguistica* rispettivamente, atte a rappresentare gran parte degli aspetti linguistici della lingua italiana.

Tuttavia, essendo la progettazione ancora in fase embrionale, l’ontologia non risultava ancora orientata ad uno specifico task.

Una volta individuato quest’ultimo, abbiamo deciso di specializzare la modellazione dell’ontologia in direzione degli aspetti sia fonetici e morfologici sia delle unità linguistiche basilari; così facendo, abbiamo reso “flat” gli aspetti sintattici e semantici in quanto di complessità maggiori rispetto ai precedenti aspetti trattati e non in linea con il task da noi scelto.

Il fine principale con cui l’ontologia è stata modellata è quello di fornire un mezzo con lo scopo di mettere in relazione i vocaboli della lingua italiana con i loro sinonimi.

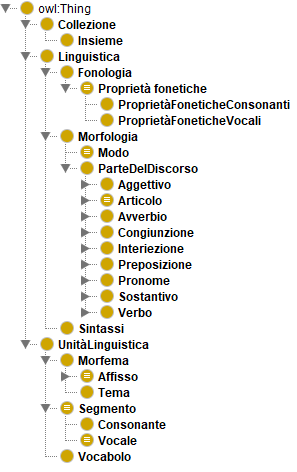
Tale tipo di task può essere visto puramente come un task di consultazione rivolto ad un ampia fetta di possibili utenti: a partire dall’allievo delle scuole primarie, all’allievo delle scuole secondarie e così via.

# 

# **Documentazione**

## **T-box**

LinSyn risulta essere composta da 67 classi, la cui organizzazione viene riportata in figura sottostante.



La classe *Collezione* e la sua sottoclasse diretta *Insieme*, modellata secondo l’Ontology Design Pattern ‘Set’ che descrive come gli elementi di un insieme devono essere relazionati tra di loro e come questi ultimi devono essere messi in relazione con la collezione che li contiene, vengono utilizzate in un contesto che è estraneo alla modellazione di tutto ciò che riguarda la parte linguistica da noi trattata.

Tali classi, difatti, vengono sfruttate nella modellazione delle collezioni, o insiemi, utili a mettere in relazione i vocaboli con i loro sinonimi.

Per quanto riguarda la modellazione degli aspetti appartenenti al contesto linguistico, si è deciso di utilizzare la classe *Linguistica* e la classe *UnitàLinguistica* che rappresentano le fondamenta dell’ontologia.

All’interno di *Linguistica* ritroviamo tutto ciò che fa parte degli aspetti fonetici, morfologici e sintattici della lingua italiana; questi ultimi sono mappati nelle classi *Fonologia*, *Morfologia* e *Sintassi* rispettivamente.

La classe *Fonologia* si occupa della rappresentazione delle proprietà fonetiche delle consonanti e delle vocali, elementi basilari e minimi delle varie lingue.

Tali proprietà sono modellate attraverso la classe *ProprietàFonetiche* che a sua volta si suddivide in *ProprietàFoneticheConsonanti* e *ProprietàFoneticheVocali*.

Nella classe *Morfologia* troviamo rappresentate le classi *Modo* e *ParteDelDiscorso*.

La prima, modellata come classe enumerata grazie al costrutto ‘Equivalent to’, viene utilizzata per rappresentare i modi verbali tra cui *condizionale*, *congiuntivo*, *gerundio*, *imperativo*, *indicativo*, *infinito* e *participio*.

La classe *ParteDelDiscorso*, invece, viene utilizzata per rappresentare tutte le varie parti del discorso tra le quali troviamo le classi:

* *Aggettivo* contenente la macro suddivisione in *AggettivoDeterminativo*, all’interno della quale troviamo le classi *AggettivoEsclamativo*, *AggettivoIndefinito*, *AggettivoInterrogativo AggettivoPossessivo,* e *AggettivoQualificativo*
* *Articolo*, classe modellata tramite restrizione utilizzando l’object property *èCostituito*, contenente le classi *ArticoloDeterminativo*, *ArticoloIndeterminativo* e *ArticoloPartitivo* utilizzata per la rappresentazione degli articoli partitivi tramite una regola SWRL
* *Avverbio* contenente le classi *AvverbioDiLuogo*, *AvverbioDiModo*, *AvverbioDiQuantità* e *AvverbioDiTempo*
* *Congiunzione* contenente una semplice suddivisione in *CongiunzioneSemplice* e *CongiunzioneComposta*
* *Interiezione* contenente le classi *InteriezionePropria*, *InteriezioneImpropria* e *LocuzioneInteriettiva*
* *Preposizione* contenente, così come *Congiunzione*, una suddivisione in *PreposizioneSemplice* e *PreposizioneArticolata*
* *Pronome* contenente al suo interno le classi *PronomeEsclamativo*, *PronomeIndefinito*, *PronomeInterrogativo*, *PronomePersonale*, *PronomePossessivo* e *PronomeRelativo*
* *Sostantivo* contenente le classi *SostantivoAstratto*, *SostantivoConcreto*, *SostantivoIndividuale SostantivoCollettivo*, *SostantivoComune* e *SostantivoProprio*
* *Verbo* contenente la suddivisione in *VerboAusiliare*, all’interno del quale troviamo la classe *VerboModale*, *VerboCopulativo* e *VerboPredicativo*

Infine, all’interno della classe *Sintassi* non vi sono ulteriori specializzazioni in quanto non in linea con il task da noi scelto.

All’interno di *UnitàLinguistica* troviamo modellati tutti quegli elementi minimali considerabili all’interno dei molteplici aspetti della linguistica. Essi sono rappresentati dalle classi *Morfema*, *Segmento* e *Vocabolo*.

Nella classe *Morfema* sono contenute le classi *Affisso*, modellata tramite l’utilizzo di restrizione utilizzando l’object property *MorfemaèCompostaDaSegmento* e contenente, a sua volta, le classi *Infisso*, *Prefisso* e *Suffisso* rappresentanti i rispettivi elementi linguistici, e *Tema*.

Nella classe *Segmento* troviamo due degli elementi minimali della linguistica rappresentati dalle classi *Consonante* e *Vocale* le quali, tramite il costrutto ‘Disjoint Union Of’ ci consentono di definire la classe *Segmento* nel seguente modo:

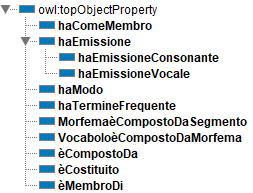
imponendo così che ciascuna istanza appartenente alla classe *Segmento* debba far parte o della classe *Consonante* o della classe *Vocale*.

Per completezza, riportiamo che la classe *Vocale* viene modellata utilizzando il costrutto ‘Equivalent to’ utilizzando la restrizione con l’object property *haEmissione* con valore *only ProprietàFoneticheVocali*.

Infine, la classe *Vocabolo* viene utilizzata per modellare i vocaboli della lingua italiana i quali vengono rapportati tra di loro attraverso l’utilizzo di una relazione generica di sinonimia.

## **Object property**

Le object property utilizzate per modellare il tema da noi trattato sono mostrate nella figura sottostante.



Per il contesto extralinguistico le object property utilizzate sono *èMembroDi* e *haComeMembro*, entrambe proprietà inverse l’una dell’altra, atte a rappresentare l’appartenenza degli elementi della classe *Vocabolo* ad una *Collezione* e viceversa.

Un ulteriore proprietà per il contesto extralinguistico è *haTermineFrequente* che ci consente di assegnare a ciascun insieme di sinonimi l’elemento più frequente di tale insieme sulla base del numero di occorrenze del vocabolo all’interno di CoLFIS.

Tra le altre proprietà utilizzate troviamo:

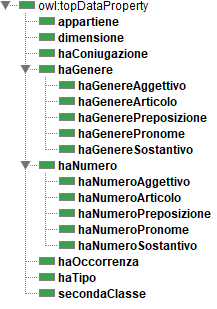
* *haEmissione* creata per mettere in relazione gli individui della classe Segmento con gli individui delle proprietà fonetiche. Tale proprietà, a sua volta, si suddivide in *haEmissioneConsonante* e *haEmissioneVocale*
* *haModo* utilizzata per relazionare le istanze della classe *Verbo* con le istanze della classe *Modo*. Così facendo è possibile associare ad ogni verbo il suo modo verbale.
* *èCompostoDa* rappresenta un esempio di property chain, in quanto viene specificata dalle property *VocaboloèCompostoDaMorfema* che lega un vocabolo a dei segmenti e da *MorfemaèCompostaDaSegmento* che lega un morfema a uno o più segmenti.

Pertanto, risulta quindi che, tramite reasoning, un vocabolo è composto da uno o più segmenti avendo così come risultato dell’inferenza: *Vocabolo èCompostoDa Segmento*.

* èCostituito utilizzata per relazionare le istanze della classe *Articolo* con le istanze della classe *Segmento* così da poter modellare la composizione di un articolo tramite dei segmenti

## **Data property**

Le data property utilizzate per modellare il tema da noi trattato sono mostrate nella figura sottostante.



Come data property, abbiamo:

* *haOccorrenza* che associa ad ogni individuo appartenente alla classe *Vocabolo* la sua occorrenza all’interno della banca dati lessicale CoLFIS
* *appartiene* e *secondaClasse* entrambe utilizzate all’interno di regole SWRL per la rispettiva associazione della classe *ParteDelDiscorso* con i valori ‘aperta’ o ‘chiusa’ e della classe *AggettivoQualificativo* con i valori ‘sì’ o ‘no’.
* *dimensione* utile per definire la dimensione di un individuo della classe *Collezione*
* *haConiugazione* che stabilisce a quale tra le tre coniugazioni della lingua italiana uno specifico verbo appartiene
* *haTipo* che stabilisce se le istanze appartenenti alla classe *Congiunzione* siano di tipo coordinativo o subordinativo
* *haGenere* che, per gli individui appartenenti alle classi *Aggettivo, Articolo, Preposizione, Pronome Sostantivo*, definisce il loro genere
* *haNumero* che, per gli individui appartenenti alle classi *Aggettivo, Articolo, Preposizione, Pronome Sostantivo*, definisce il loro numero

## **A-box**

All’interno della parte asserzionale di LinSyn sono stati inseriti circa un centinaio di individui, di cui ne mostriamo una parte in figura sottostante.



Al suo interno possiamo trovare le vocali e le consonanti della lingua italiana, aggettivi di vario tipo, articoli determinativi e indeterminativi, esempi di avverbi, congiunzioni composte e semplici, esempi di interiezioni e di locuzioni interiettive, preposizioni, pronomi, sostantivi ed infine verbi.

Inoltre, sono anche asseriti, a scopo di testing, due individui nominati *sinonimiBello* e *sinonimiPrimo* i quali contengono rispettivamente i sinonimi dei vocaboli ‘bello’ e ‘primo’.

Tali insiemi vengono utilizzati per elencare tutte le possibilità che fanno fronte alle esigenze di un ipotetico utente nel dover scegliere i sinonimi di uno o più vocaboli.

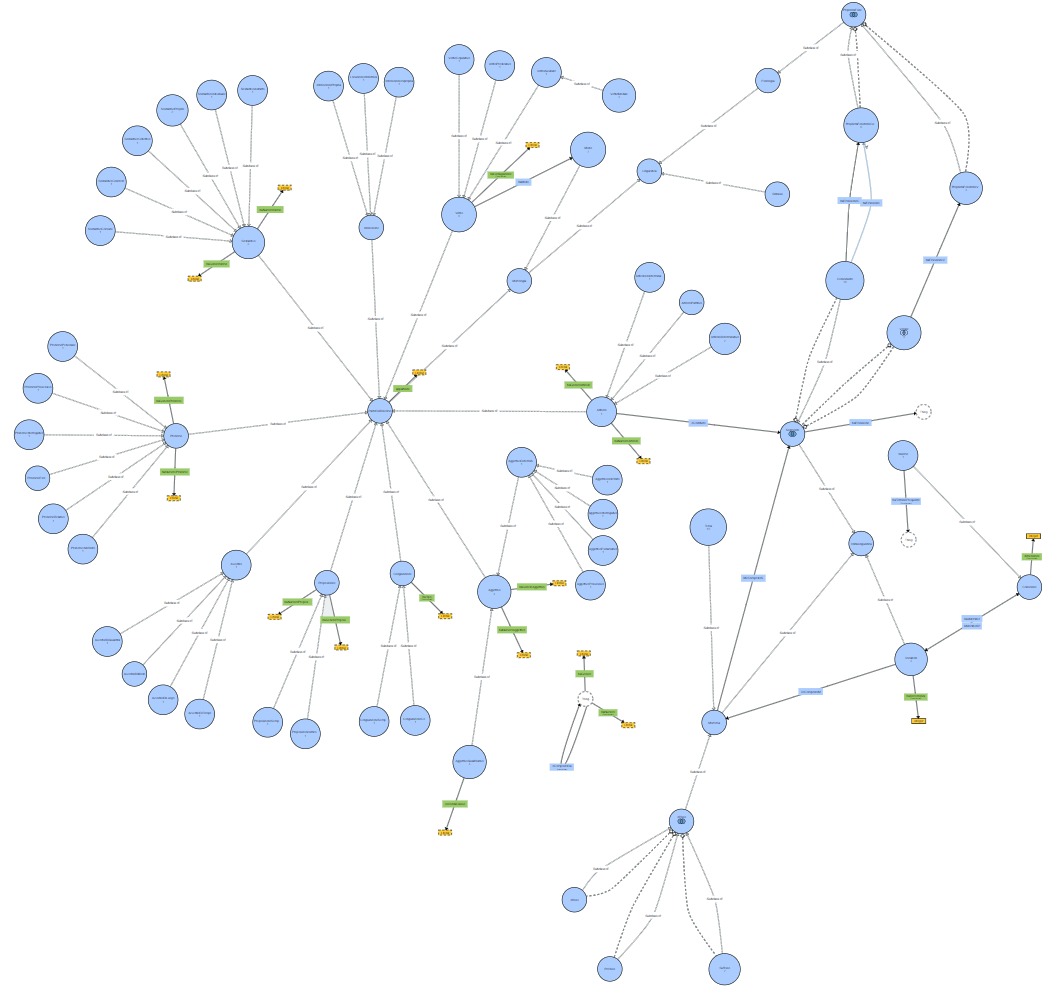
A tali insiemi è associato il termine più frequente, ovvero il termine facente parte di tale insieme che presenta occorrenza massima in segutio all’analisi della banca dati CoLFIS.

(riscrivi LOL)

Tale metrica potrebbe essere utilizzata per effettuare delle raccomandazioni ad un possibile utente in cerca di sinonimi da utilizzare. Sulla base di questa misura, tramite un’operazione di ranking, sarà possibile ordinare tutti gli elementi presenti all’interno di un insieme e associare all’elemento con il ranking più alto la maggior importanza.

# 

# **Visualizzazione dell’ontologia**



# **Query SPARQL**

Come richiesto dalle specifiche, abbiamo provveduto alla creazione di una decina di query SPARQL atte all’interrogazione dell’ontologia LinSyn.

La totalità delle query è stata pensata per effettuare un’interrogazione più ampia e generica possibile degli elementi presenti all’interno di LinSyn.

NB: Riportiamo, nell’elenco sottostante, i prefissi utilizzati nelle query SPARQL:

- PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

- PREFIX owl: <http://www.w3.org/2002/07/owl#>

- PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

- PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

- PREFIX p: <http://www.semanticweb.org/giacomo/ontologies/2019/2/LinSyn#>

Inoltre, riportiamo di seguito le query svolte:

1. “Tutti i sostantivi maschili o femminili a scelta.”

SELECT ?sostantivo

WHERE {

?sostantivo rdf:type ?type.

?sostantivo p:haGenereNome ?haGenere.

FILTER( regex(str(?o1), "Sostantivo") && regex(str(?haGenere), queryParam = (“M/”F”) ) ) }

1. “Tutte le interiezioni.”

SELECT DISTINCT ?interiezione

WHERE {

?interiezione rdf:type ?type

FILTER (CONTAINS(str(?type), "Interie")) }

1. “Tutti i vocaboli con tema in comune 'bell', 'splendid', ecc.”

SELECT ?vocabolo

WHERE {

?vocabolo rdfs:label ?label

FILTER( regex(str(?label), '^" + queryParam = (“bell”, “splendid”, … ) + "') ) }

1. “Tutte le proprietà fonetiche sia consonanti che vocali.”

SELECT ?proprietàFonetiche

WHERE {

?proprietàFonetiche rdf:type ?s1 {

SELECT ?s1

WHERE {

?s1 rdfs:subClassOf ?subject {

SELECT ?subject

WHERE { \n"

?subject rdfs:subClassOf ?s2 {

SELECT ?s2

WHERE {

?s2 rdfs:subClassOf ?o1

FILTER(regex(str(?s2),"Fonologia")) } } } } } }

1. “Tutte le consonanti con la loro emissione.”

SELECT ?consonanti ?emissione

WHERE {

?consonanti rdf:type ?type.

?consonanti p:haEmissioneC ?emissione

FILTER( regex( str(?type), "Consonante" ) ) }

ORDER BY (?consonanti)

1. “Tutti i nomi propri.”

SELECT DISTINCT ?nomeProprio

WHERE {

?nomeProprio rdf:type ?type {

SELECT ?type

WHERE {

?subj rdf:type ?type }

}

FILTER( regex( str(?type), “SostantivoProprio" ) ) }

1. “Elencare le parti del discorso.”

NB: Riportiamo, nell’elenco sottostante, i prefissi utilizzati nelle query SPARQL:

SELECT ?POS

WHERE {

?POS rdfs:subClassOf ?subClass

FILTER( regex( str(?subClass), "ParteDelDiscorso” ) ) }

1. “Tutti i modi dei verbi.”

NB: Riportiamo, nell’elenco sottostante, i prefissi utilizzati nelle query SPARQL:

SELECT ?modoVerbale

WHERE {

?modoVerbale rdf:type ?type

FILTER(regex(str(?type), "Modo"))}

1. “Dato un insieme di sinonimi, elencare il TermineFreqente con l'uso della data property.”

NB: Riportiamo, nell’elenco sottostante, i prefissi utilizzati nelle query SPARQL:

SELECT ( max(?occurencies) as ?maxOccur )

WHERE {

?insiemeSinonimi rdf:type ?c1.

?elem p:isMemberOf ?insiemeSinonimi.

?elem p:haOccorrenza ?occurencies

filter( regex( str(?c1), "Insieme") "

&& regex(str(?insiemeSinonimi), '" + queryParam = (sinonimiBello, sinonimiPrimo, ...) + "'))}

Inoltre, un’ulteriore richiesta prevedeva l’integrazione di due query da effettuare su di un endpoint SPARQL pubblico a scelta tra DBpedia, Wikidata, ecc

Le due query, effettuate su Wikidata, vengono riportate qui di seguito:

1. “Sottoclassi delle proprietà fonetiche delle consonanti (Wikidata):”

SELECT ?itemLabel

WHERE

{

?item wdt:P279 wd:Q38035.

SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO\_LANGUAGE],en". }

}

1. “Sottoclassi della linguistica (Wikidata).”

SELECT ?itemLabel

WHERE

{

?item wdt:P279 wd:Q8162.

SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO\_LANGUAGE],en". }

}

# 

# **Regole SWRL**

La base di regole SWRL da noi creata risulta essere composta nel seguente modo:

1. “Aggettivi qualificativi della II°classe, quelli che, al singolare e al plurale, concordano per entrambi i generi.”

progettomodsem:AggettivoQualificativo(?x) ^ progettomodsem:haGenere(?x, ?y1) ^ progettomodsem:haGenere(?x, ?y2) -> progettomodsem:secondaClasse(?x, "sì")

1. “Articoli partitivi”

progettomodsem:Articolo(?x) ^ swrlb:containsIgnoreCase(?x, "de") -> progettomodsem:ArticoloPartitivo(?x) (ok)

progettomodsem:Articolo(?x) ^ swrlb:matches(?x, "(de)") -> progettomodsem:ArticoloPartitivo(?x) (no, confronto l'URI con una stringa)

progettomodsem:Articolo(?x) ^ swrlb:startsWith(?x, "(de)") -> progettomodsem:ArticoloPartitivo(?x) (no, confronto l'URI con una stringa)

1. “Classe aperta” e “Classe chiusa” per le Parti del Discorso (POS)”

progettomodsem:SostantivoProprio(?x) -> progettomodsem:appartiene(?x, "aperta")

progettomodsem:Preposizione(?x) -> progettomodsem:appartiene(?x, "chiusa")

1. “Tutte le Parti del Discorso (POS) contenenti "?" riclassificate”

progettomodsem:Pronome(?x) ^ swrlb:endsWith(?x, "'?") -> progettomodsem:PronomeInterrogativo(?x) (no, confronto l'URI con una stringa) – mettere etichetta

progettomodsem:Pronome(?x) ^ swrlb:containsIgnoreCase(?x, "?") -> progettomodsem:PronomeInterrogativo(?x)

Inoltre, a scopi puramente didattici, abbiamo prodotto un’ulteriore regola andando, però, ad utilizzare SQWRL (Semantic Query-Enhanced Web Rule Language), che riportiamo qui sotto:

1. “Tutti i vocaboli con la proprietà haOccorrenza ed ordinamento descrescente in base alla loro occorrenza “

Vocabolo(?x) ^ haOccorrenza(?x, ?occ) . sqwrl:makeSet(?sv, ?occ) -> sqwrl:select(?x, ?occ) ^ sqwrl:orderByDescending(?occ)

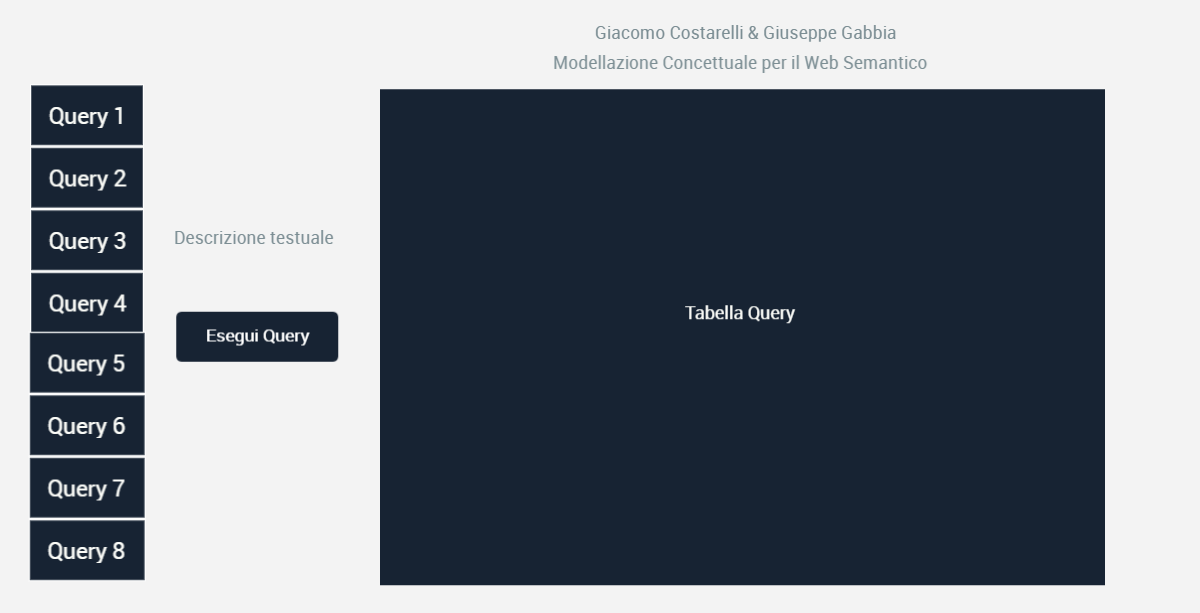
# 

# **Applicazione client**

Infine, utilizzando una Linked Data Platform (Virtuoso, GraphDB o altri), è stata effettuata l’integrazione delle query SPARQL precedentemente riportate all’interno di un’applicazione client.

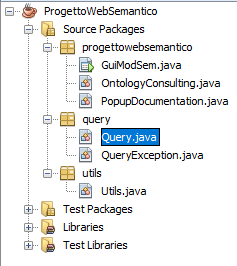
La scelta della Linked Data Platform da utilizzare è ricaduta su GraphDB. Una volta effettuata la configurazione di quest’ultimo applicativo ed aver esportato l’ontologia LinSyn nel formato opportuno, abbiamo provveduto alla creazione di una repository, nominata *my\_repo*, all’interno della quale è stato effettuato l’upload dei nostri dati.

Una volta fatto ciò, abbiamo iniziato a progettare l’interfaccia della nostra applicazione client tramite dei mockup creati con l’ausilio di FluidUI, reperibile all’indirizzo [*https://www.fluidui.com/*](https://www.fluidui.com/) e che riportiamo di seguito:



Completata l’attività di progettazione, abbiamo provveduto alla codifica dell’applicazione client in linguaggio Java utilizzando il framework Swing.

L’organizzazione del progetto, nei vari package, risulta essere così composta:



Partendo dal basso troviamo il package *utils* che contiene la classe *Utils.java* contenente:

* *setUpConnection*() utilizzato per effettuare la connessione con la repository GraphDB *my\_repo*
* *makeQuery*() utilizzato per effettuare una delle query SPARQL presentate in precedenza
* *fillTable*() utilizzato per estrarre i dati risultanti sia da una query sia da un file CSV contenente i risultati di una query SPARQL.
* *readCSV*() utilizzato per leggere i dati contenuti in un file CSV
* *getIPAddress*() utilizzato per recuperare, in modo dinamico, l’indirizzo IP della macchina su cui è attiva un’istanza di GraphDB

Salendo troviamo il package *query* contenente la classe *Query.java* contenente il metodo *getQuery*() atto a creare la query da eseguire, dati il nome del bottone ed il parametro, opzionale per alcune query, selezionato dall’utente e la classe *QueryException.java* utile nel caso si vogliamo produrre eccezioni ad-hoc per query errate.

Nel package *progettowebsemantico* troviamo le classi *GuiModSem.java*, *OntologyConsulting.java* e *PopupDocumentation.java*.

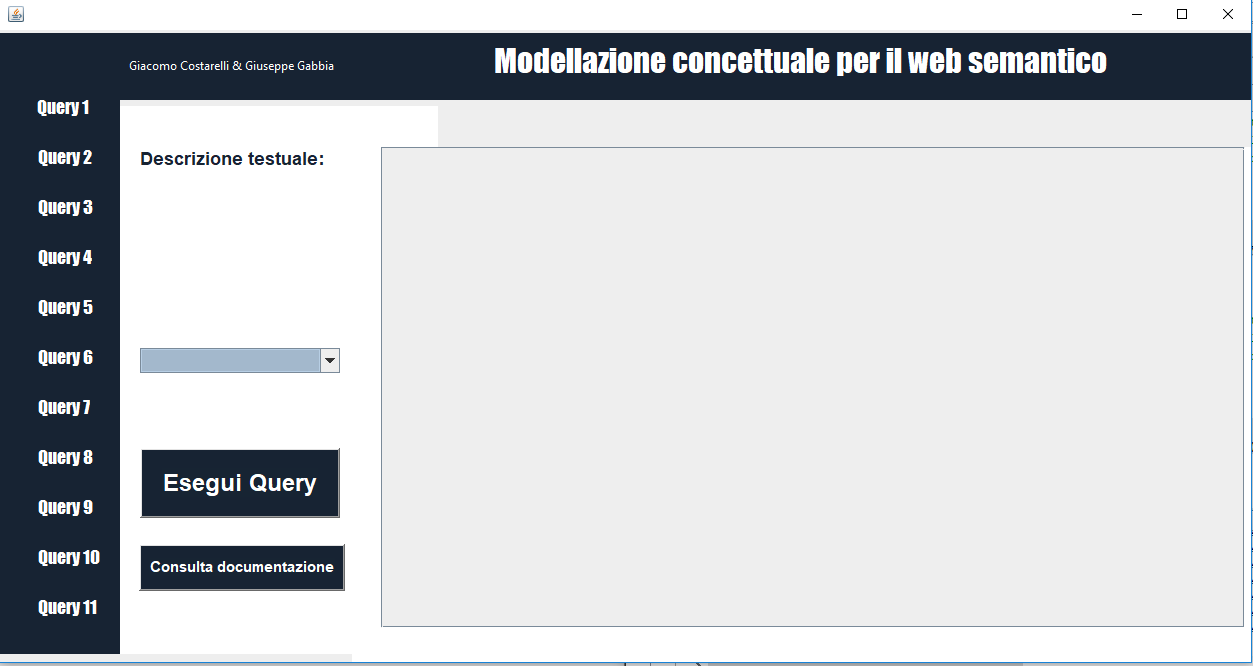
La classe *GuiModSem.java*, classe e frame principale del progetto, si compone di un numero di *buttonQuery* pari al numero di query SPARQL integrate al suo interno, di un *runQueryButton* utilizzato per effettuare l’interrogazione dell’ontologia, il processamento dei dati risultanti da quest’ultima e la visualizzazione all’interno dell’interfaccia dell’applicazione.

La classe OntologyConsulting.java definisce la struttura di una nuova finestra che viene mostrata nel caso in cui l’utente selezioni attraverso il pulsante “Consulta la documentazione” presente nella finestra principale GuiModSem.

Questo frame ci mostra attraverso una lista, implementata con una JList, contenente le principali classi rappresentate nell’ontologia, permettendo all’utilizzatore di selezionare una qualsiasi di esse al fine di avere un riscontro testuale che descriva gli elementi elencati. Da questa finestra è possibile effettuare un doppio click su uno qualsiasi degli elementi della lista per accedere all’ultima classe componente la nostra GUI: PopupDocumentation.java.

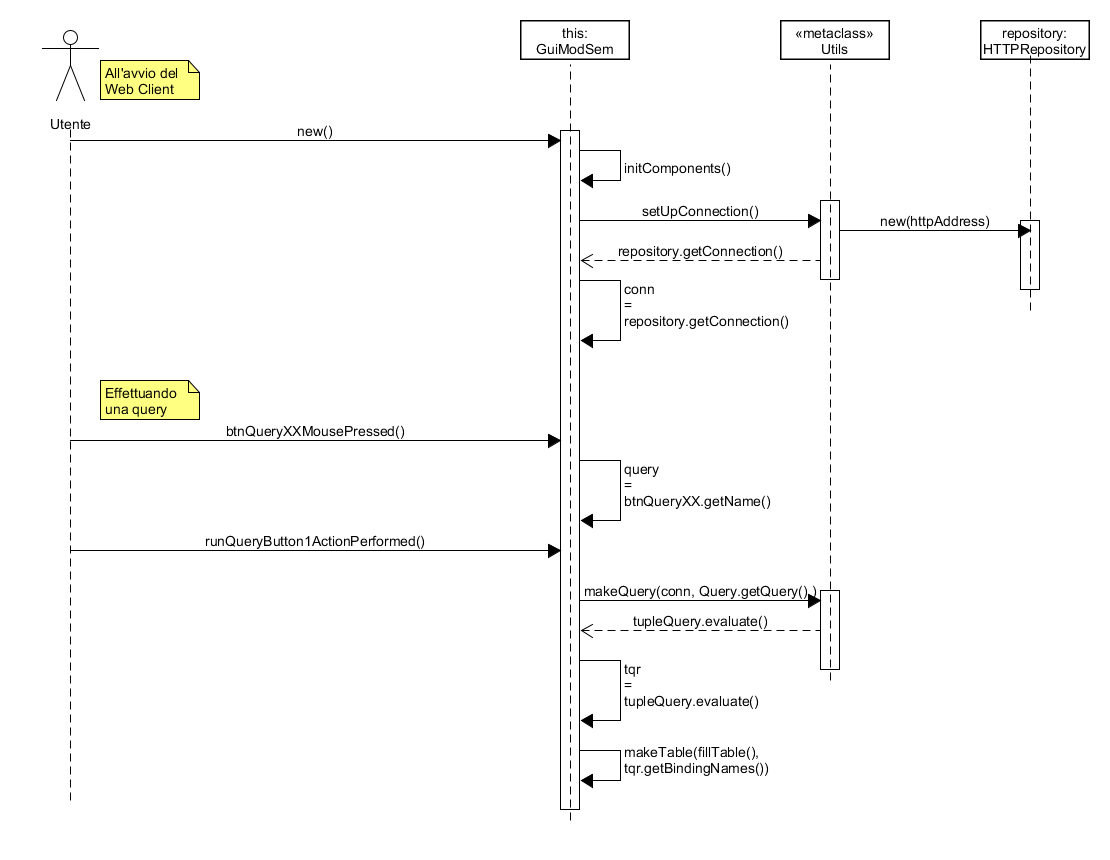
Quest’ultima classe è composta da un JFrame contenente una JTextArea che mostra all’ utente la descrizione di cui prima attraverso la selezione dell’elemento di cui si vuole ottenere la descrizione.

Graficamente l’applicazione client si presenta nel seguente modo:



L’interazione con l’utente avviene mediante i vari *button* presenti all’interno dell’interfaccia i quali permettono di scegliere una tra le varie query SPARQL da eseguire.

L’interazione *Utente-Query* può essere generalizzata e schematizzata nel modo seguente:

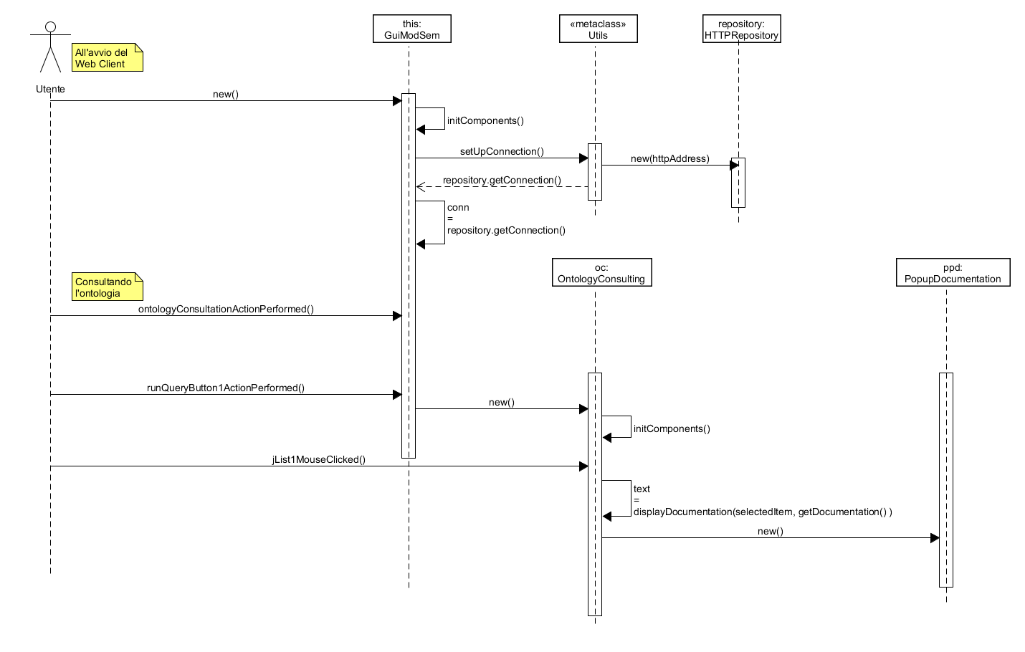


È inoltre possibile scegliere, per alcune query, un parametro opzionale grazie all’utilizzo di una *JComboBox*, a cui si fa riferimento all’interno delle varie interrogazioni con il nome di *queryParam*, così da evitare codice verboso e lasciare una libertà, seppur minima, all’utente in fase di interrogazione dell’ontologia.

Ulteriore funzione che è stata implementata è quella di consultazione dell’ontologia che viene supervisionata dalla classe OntologyConsulting.java. La finestra descritta da questa classe viene mostrata quando l’utente seleziona il JButton con campo text “Consulta l’ontologia” il cui evento MouseClicked viene catturato dall’handler ontologyConsultingActionPerformed.

La GUI esporrà quindi il frame descritto da OntologyConsulting.java.

L’interazione di questo modulo, invece, può essere schematizzata nel modo seguente:



Infine, riportiamo per completezza anche uno snapshot dell’applicazione client una volta processata in toto un’interrogazione tra quelle disponibili:

