Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

JLO - Json Layout for querying Ontology

Tesi di Laurea in Informatica

Relatore:
Prof. Paolo Ciancarini
Correlatore:
Dott. Francesco Poggi

Presentata da:
Luca Bonini

FEB, 2017

Sommario

L'obiettivo di questa tesi è introdurre un linguaggio per la rappresentazione grafica dei dati semantici, chiamato *Json Layout for quering Ontology* (JLO). In particolare, JLO si occupa di fornire tutti i dati necessari per produrre un grafo ad uno script implementato con D3.js [1]. Nel primo capitolo presenteremo brevemente il Semantic Web, la sua architettura e i principi. Ci concentreremo poi sul concetto di *Abox* e di *Tbox* mostrandone le differenze. Infine ragioneremo sull'importanza di rappresentare i dati graficamente mostrando degli esempi di software che svolgono questo compito in vari contesti. Nel secondo capitolo faremo luce sullo stato dell'arte attuale, indagando quali strumenti sono stati sviluppati per risolvere problemi simili al nostro, quali le loro applicazioni e quali i loro limiti. In particolare divideremo i tool in due categorie: gli strumenti per la visualizzazione di dati semantici "per utenti esperti" e "per tutti gli utenti". Mostreremo inoltre nel dettaglio alcuni di questi come , "RDF Gravity", "MEMOgraph" e "WebVOWL". Nel terzo capitolo invece si parlerà di JLO: inizieremo spiegando quali sono le sue fondamenta e quali le tecnologie utilizzate. Successivamente esibiremo il progetto di una web app che si occuperà di supportare la comprensione del contenuto di grandi basi di conoscenza in formato semantico. Questo viene realizzato in quattro fasi:

1. *Reperire i dati dalla base di conoscenza*, per permettere la scelta del dataset da interrogare e i dati che si vogliono ottenere con una query sparql.

[1] https://d3js.org

2. *Creazione delle regole di presentazione*, per consentire all'utente tramite un interfaccia web interattiva di poter stabilire come verrà disegnato il grafo dei risultati.

3. *Lo sviluppo di un file in formato JLO.*

4. *La creazione della visualizzazione*, tramite uno script che prende in input un file JLO e che ne mostra i risultati in una scheda del browser.

Illustreremo inoltre una possibile applicazione di questo tool che prevede la visualizzazione personalizzata di dati estratti da DBpedia.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione: il contesto

1.1 Il Semantic Web

Il "Semantic Web", così chiamato dal suo ideatore, Tim Berners-Lee, è un insieme di tecnologie che ha come scopo quello di rappresentare il web, in un formato comprensibile dalle macchine, intese come infrastrutture e applicazioni software. L'obbiettivo è di consentire a queste di produrre nuova conoscenza ricavando delle conclusioni partendo da una conoscenza di base iniziale. Il Semantic Web va inteso come una struttura "a strati" oppure, per usare un termine informatico "a stack": esso è formato da più componenti in cui ogni livello è la base per gli standard definiti a livello superiore come si può vedere in figura 1.1 [1]. In questo documento tratteremo argomenti che si posizionano sul livello delle ontologie.

L'evoluzione del web in Semantic Web inizia con la definizione, da parte del W3C, dello standard Resource Description Framework (RDF), una particolare applicazione XML che standardizza la definizione di relazioni tra informazioni ispirandosi ai principi della logica dei predicati (o logica predicativa del primo ordine) e ricorrendo agli strumenti tipici del web (ad es. URI) e

[1] https://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html

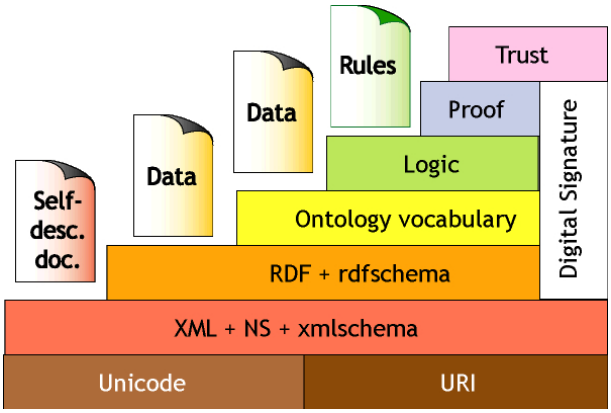

Figura 1.1: Struttura Semantic Web

dell'XML (namespace). Secondo la logica dei predicati le informazioni sono esprimibili con asserzioni costituite da triple formate da soggetto, predicato e oggetto. Ad esempio, le seguenti affermazioni sulla relazione di due persone all'interno di un Social Network:
Io (Luca) sono amico di Silvia.

	Asserzione
Soggetto	*Luca*
Predicato	*amico di*
Oggetto	*Silvia*

Tabella 1.1: Tabella esempio Tripla RDF

Vediamo degli esempi di fantasia di come potrebbero essere identificati questi elementi:

- Luca
  http://www.facebook.com/Luca
- Amico di
  http://it.wiktionary.org/wiki/amicodi

- Silvia
  http://www.facebook.com/Silvia

ABox e TBox

Un'ontologia tipicamente consiste in due parti. La parte terminologica introduce i concetti e le proprietà dandogli una struttura usando i costrutti disponibili dal linguaggio (chiamato per questo TBox nelle descrizioni logiche). L'altra parte, quella che contiene l'asserzione,fornisce gli individui e le relazioni tra di loro. Per esempio, consideriamo un'ontologia di un Social Network dove la Tbox definisce i termini come Persona (definita per esempio come unione di sottoinsiemi quali *Male* e *Female*) o Azienda, così come proprietà del tipo *has-friend*, *supervisorOf*,*has-Child* e *worksWith*. La corrispondente ABox copre tutti gli individui del dominio, come per esempio *Pietro*, *Monica*, *Silvia* e proprietà come *Pietro has-firend Silvia*. Un ampio Social Network è caratterizzato da un ampio numero di utenti e di proprietà che intercorrono tra di loro. Infatti in un applicazione nel mondo reale, le TBox ammontano ad un numero minore rispetto alle asserzioni contenute nel ABox [8].

1.2 Il problema di rappresentare i dati

L'evoluzione della tecnologia hardware consente ai sistemi automatizzati del giorno d'oggi di memorizzare un grande quantitativo di informazioni. I ricercatori dell'Università di Berkeley stimano che, ogni anno, più di un Exabyte (1 Milione di Terabyte) di dati vengono generati [4]. Questo significa che in futuro, con una continua evoluzione della tecnologia, verrà generato un quantitativo di dati sempre maggiore. Il Semantic Web, per esempio, dal 2009 ad oggi ha avuto un espansione notevole che possiamo constatare dalle seguenti immagini che raffigurano i dataset presenti:

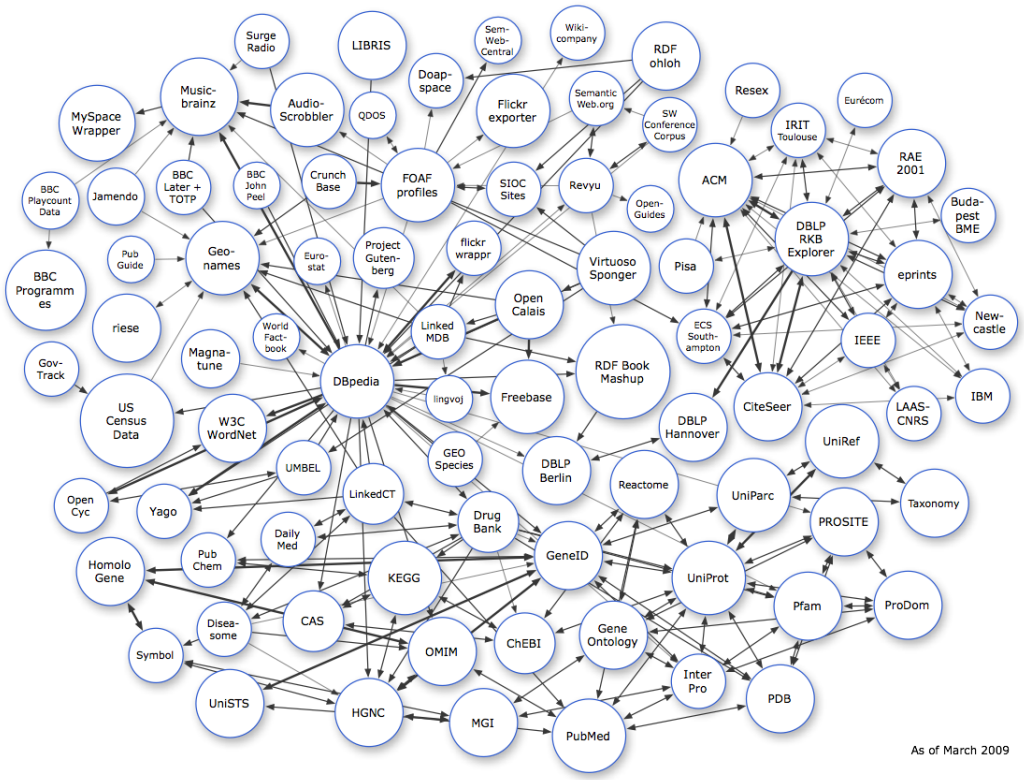

Figura 1.2: Dataset presenti nel 2009

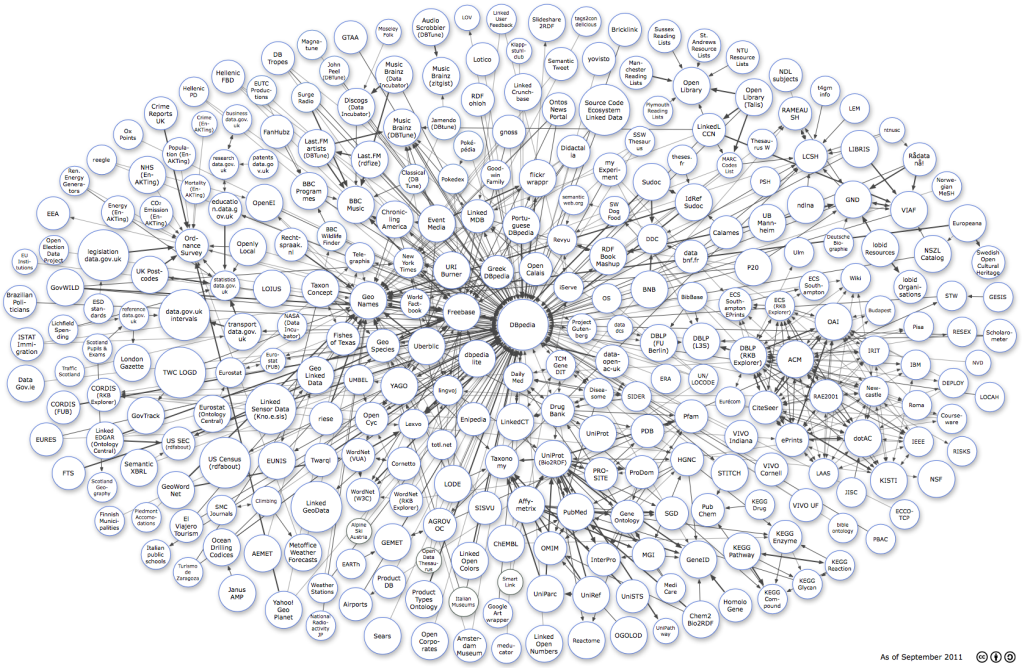

Figura 1.3: Dataset presenti nel 2011

I dati vengono memorizzati poiché crediamo essere una fonte di informazioni da cui trarre dei vantaggi. Questa crescita del web-of-data rende difficile

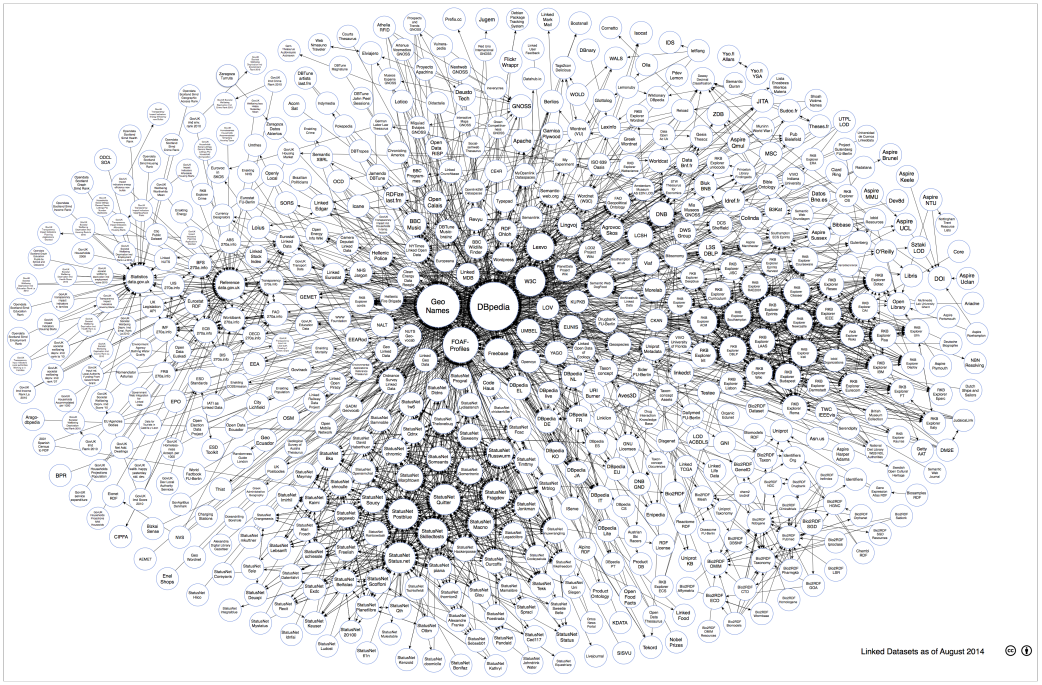

Figura 1.4: Dataset presenti nel 2014

la comprensione di queste informazioni e di conseguenza lo è anche il suo utilizzo. Per questo motivo è importante riuscire a rappresentare i dati includendo l'uomo nel processo di esplorazione, combinando la sua flessibilità, creatività e la sua generale conoscenza del genere umano con l'enorme capacità di memoria e di computazione dei computer di oggi. L'idea che sta alla base dell'esplorazione visiva è quella di presentare i dati in qualche forma, che consenta all'uomo di ottenere una visione "dall'interno", trarre delle conclusioni e di interagire direttamente con la base di conoscenza. La tecnica dell'esplorazione visiva si è dimostrata essere di grande importanza nell'esplorare grandi quantità di dati. Questa può essere vista come un processo che genera una successione di ipotesi: la visualizzazione di dati porta l'utente a ottenere facilmente nuove informazioni, le quali possono portare alla formulazione di nuove ipotesi. La verifiche di quest' ultime possono attuarsi tramite l'esplorazione grafica. In aggiunta al diretto coinvolgimento dell'utente , i maggiori vantaggi dell'esplorazione visiva di informazioni, unita a

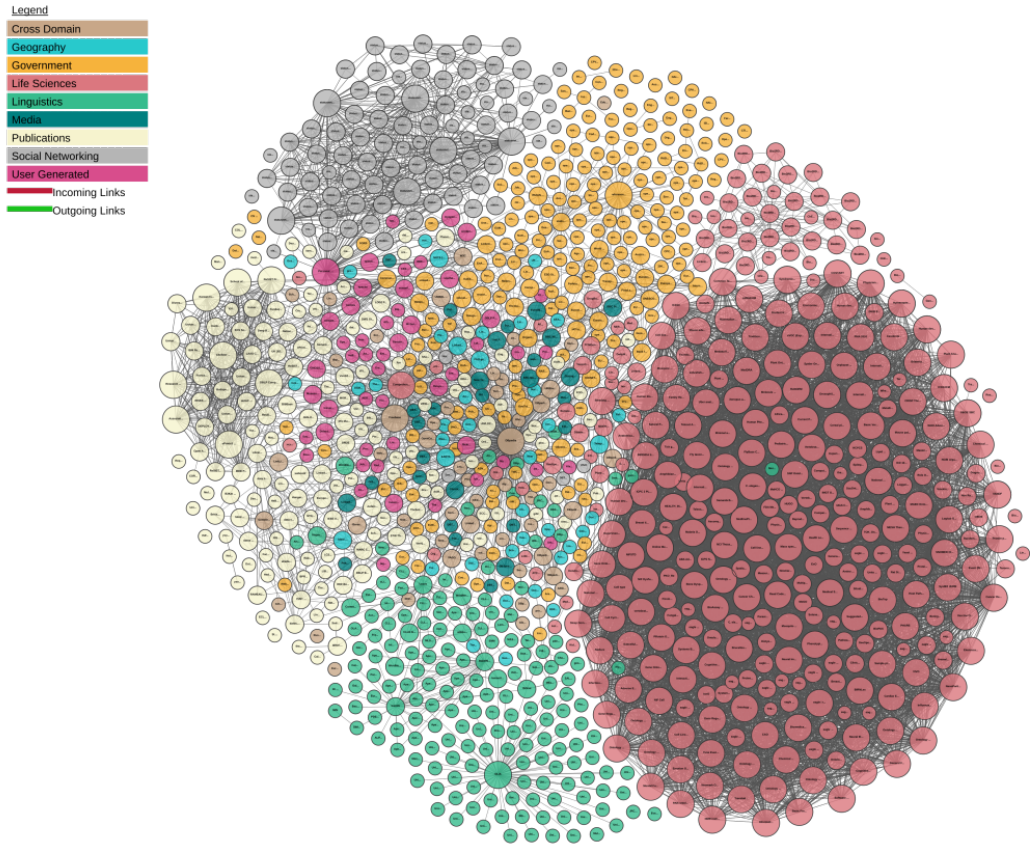

Figura 1.5: Dataset presenti nel 2017

tecniche di estrazione automatica da parte della macchina sono:

- La tecnica dell'esplorazione visiva può facilmente maneggiare un grande quantitativo di dati non omogenei.
- L'esplorazione visiva è intuitiva e non richiede una profonda conoscenza della struttura del dataset e delle relazioni che intercorrono tra i dati. Inoltre fornisce un maggiore grado di confidenza nella ricerca.

Generalmente la ricerca grafica passa attraverso 3 step: *una prima panoramica dei dati ottenuti*, un successivo *filtraggio delle informazioni*, infine una *messa a fuoco* su ciò che si ritiene significativo. Inoltre, per ottenere un efficiente esplorazione dei dati è necessaria una qualche tecnica di interazione. Questa consentirebbe all'analizzatore di interagire direttamente con la visualizzazione, cambiando dinamicamente il modo di illustrare i risultati ottenuti in base al proprio obiettivo. Il nostro scopo è quello di portare su ogni computer le potenzialità della tecniche di visualizzazione per permettere una veloce, migliore e intuitiva esplorazione di grandi quantità di dati,non solo vantaggiosa in termini economici ma anche di soddisfazione finale dell'utente. Nei capitoli successivi illustreremo il progetto di una web app per la rappresentazione di dati Semantici, ma che potrebbe essere usata anche nei contesti delle seguenti applicazioni.

1.2.1 Esempi di software per la visualizzazione di dati non Semantici

Where does my money go
http://app.wheredoesmymoneygo.org/

Rappresentare i dati graficamente può essere importante in diversi contesti tra cui l'Open Data per quanto riguarda la trasparenza: un progetto britannico come "Where does my money go" permette di identificare come i soldi delle tasse dei cittadini vengano spesi dal governo.

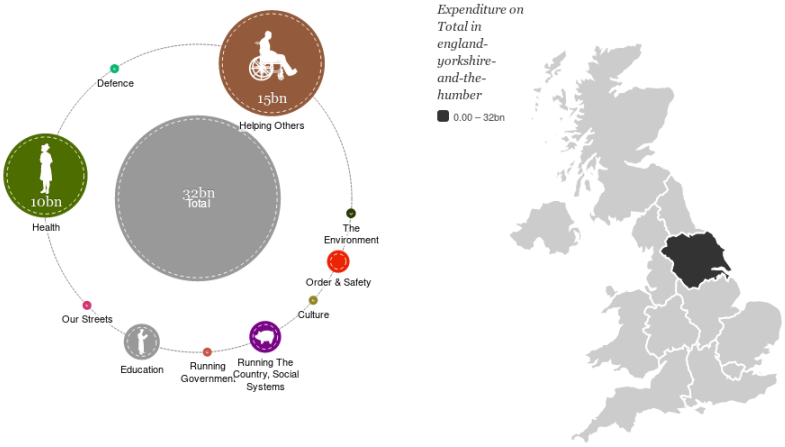

Figura 1.6: Where does my money go

GeoDa
http://geodacenter.github.io/

GeoDa è un software open source che serve come introduzione all'analisi dei dati spaziali. GeoDa è stato sviluppato dal Dr.Luc Anselin e il suo team. Il programma fornisce un interfaccia grafica "user-friendly" per l'esplorazione di dati spaziali (ESDA).

Datacomb
https://github.com/cmpolis/datacomb

Datacomb è un tool interattivo scritto da Chris Polis per l'analisi e l'esplorazione attraverso dati tabulari.

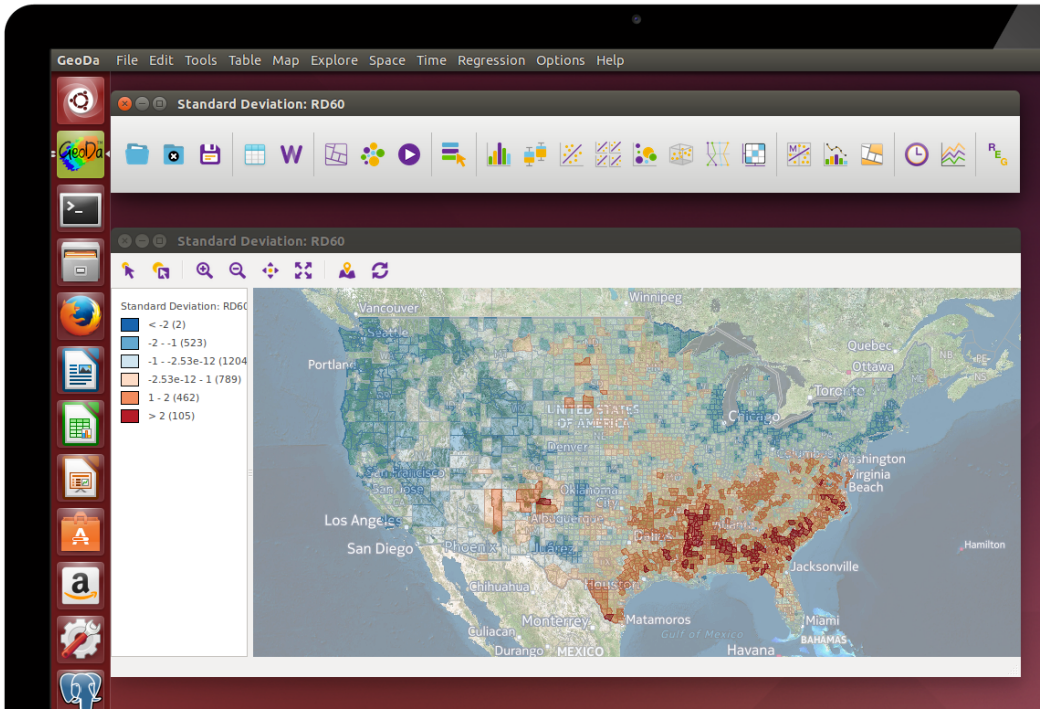

Figura 1.7: GeoDa su ubuntu

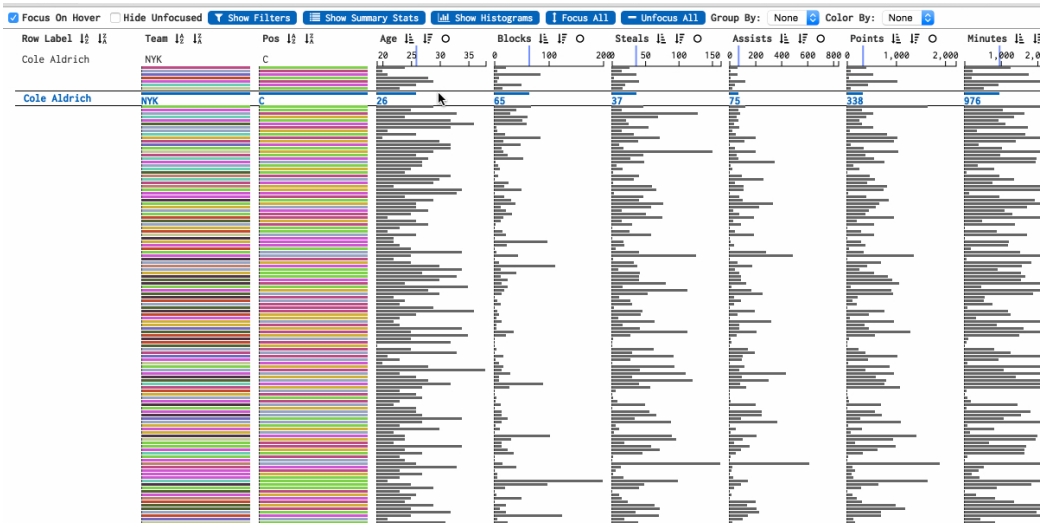

Figura 1.8: Datacomb in esecuzione

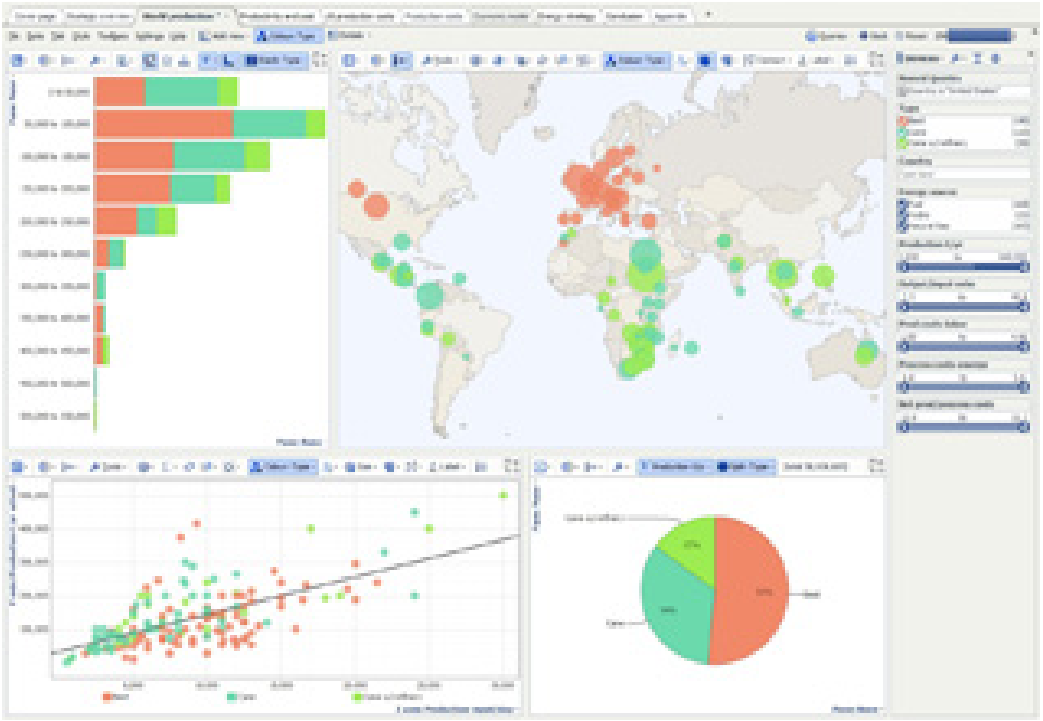

Figura 1.9: Visioko in esecuzione

Visioko Omniscope
http://www.visokio.com/

Visioko è tool per l'analisi di dati versatile, multi-tab, multi-view e interattivo. Offre un nuovo e potente modo di visualizzare,esplorare e rapportare ampie tabelle di dati-con le relative immagini, mappe, collegamenti e altro. Inoltre consente di condividere i propri file con gli altri utenti.

WordSeer
http://wordseer.berkeley.edu/

WordSeer è un ambiente per l'analisi del testo che unisce visualizzazione, recupero di informazioni ed elaborazione di linguaggio per rendere il contenuto del testo navigabile, accessibile e utile.

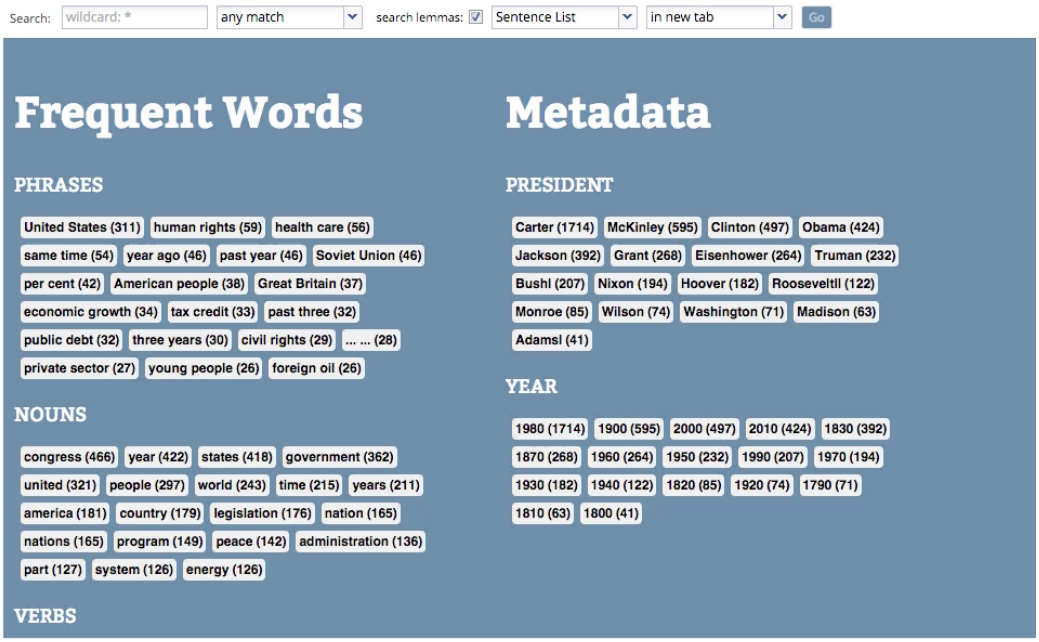

Figura 1.10: WordSeer in esecuzione

Capitolo 2

Lo stato dell'arte: strumenti per la visualizzazione di dati Semantici

In questo capitolo, presenteremo un confronto tra qualche strumento per la visualizzazione di ontologie. Classificheremo questi tool in due grandi categorie: "Strumenti per la visualizzazione di ontologie per esperti" e "Strumenti per la visualizzazione di ontologie per tutti". Illustreremo prima una tabella comprensiva dei maggiori strumenti in circolazione per poi concentrarci in particolare su due esempi per categoria. In particolare per la prima categoria presenteremo Protégé con uno dei suoi plugin più famosi ossia GrOWL, e RDF Gravity. Per la seconda categoria presenteremo MEMOgraph e WebVOWL. Ogni software verrà valutato per la capacità di visualizzazione, l'aspetto estetico, l'usabilità e la documentazione, usando una scala da 1 a 5 per ogni caratteristica.

- Capacità di visualizzazione : abilità di visualizzare gli elementi di un'ontologia. Più è alta la votazione più il software è in grado di mostrare dettagli su di essa come classi, proprietà, tipo di dato ecc.

- Usabilità : rappresenta quanto è facile usare il software.

- Aspetto estetico : un software è considerato esteticamente bello quanto più usa forme e colori per rappresentare elementi diversi o concettualmente simili.
- Documentazione : rappresenta quanto la documentazione del tool è reperibile, accurata e dettagliata.
- Personalizzazione: rappresenta il grado di personalizzazione del tool.

Dalla tabella in figura 2.1 notiamo che tutti i software sono implementati in java tranne per WebVOWL che è scritto in HTML,CSS e JavaScript, mentre Knoocks è implementato in C#. Inoltre si può vedere la grande disparità tre i software orientati ad un "utente esperto" e quelli "per tutti" [3].

2.1 Strumenti di visualizzazione per esperti

2.1.1 Protégé

Protégé è il principale editor di ontologie e knowledge management libero e open source. Protégé fornisce all'utente un interfaccia grafica utile per definire ontologie, costruire i domini, personalizzare i dati, definire classi, gerarchie e variabili. Inoltre è possibile definire le relazioni tra classi e le relative proprietà. Include inoltre un classificatore deduttivo per convalidare quali modelli sono coerenti e dedurre nuove informazioni basate sull'analisi di un'ontologia. Come Eclipse, Protégé è un freamwork, per cui supporta lo sviluppo di plugin. Questa applicazione è scritta in Java e fa un uso massiccio della libreria Swing per creare le interfacce per l'utente. Protégé conta circa 300,000 utenti registrati ed è un software molto importante in questa campo perché è la base di altri programmi che andremo a descrivere successivamente. Protégé è stato sviluppato alla Standford University ed è reso disponibile sotto la licenza BSD 2-clause license [1].

	2D/3D	Representation			Interaction			Development platform	Ontology	Availability
		Classes And instances	Object properties	Datatype properties	Filter	Search	Zoom			
"Ontology visualization tools for only ontology experts"										
Protégé class browser [2]	2d	Classes as nodes in an indented tree. Instances displayed in a separate window.	-	+	+	+	-	Java	RDF, OWL	+
OWLViz [†]	2d	Labeled colored ellipses.	-	-	+	-	+	Java	OWL	+
KC-Viz [3]	2d	Classes as labels.	+	-	+	-	+	Java	OWL	+
OntoGraf[‡]	2d	Classes as labeled rectangles with a small brown circle. Instances as labeled rectangles with a purple diamond.	+	-	+	+	+	Java	OWL	+
TGViz Tab[4]	2d	Classes and instances as labels of different colors.	+	-	+	+	+	Java	OWL	+
GrOWL[5]	2d	Classes as labeled rectangles	+	+	+	+	+	Java	OWL	-
OWLPropViz[§]	2d	Classes as labeled ellipses.	+	-	+	-	+	Java	OWL	+
SOVA[**]	2d	Classes as labeled rectangles.	+	+	+	-	+	Java	OWL	+
Jambalaya[6]	2d	Rectangles inside their parent node. Different colors for classes and instances.	+	-	+	+	+	Java	OWL	+
GLOW[7]	2d	Various layouts are available. Top class is placed in the outer ring having sub-classes placed in consecutive inner circles (inverted radial trees). Classes as labels around the central node (force-directed graph).	+	+	-	-	+	Java	OWL	+
ezOWL[8]	2d	Class as a table (name /properties).	+	+	+	-	+	Java	OWL	+
IsaViz[††]	2d	Classes and instances as labeled ellipses.	+	+		+	+	Java	RDF	+
OntoViz Tab[9]	2d	Rectangle nodes with different colors for classes and instances.	+	+	-	-	+	Java	OWL	-
OntoSphere[10]	3d	Classes and instances as spheres.	+	-	-	-	+	Java	OWL, RDF	+
Ontorama[11]	2d	Classes and instances as labels around the central node.	+	+	-	-	-	Java	RDF	-
Onto3DViz[12]	3d	Classes are represented as 3d spheres. Instances are represented as 3d boxes.	+	+	+	-	+	Java	OWL	-
NavigOWL[13]	2d	Classes and instances as circles. Different colors for classes (yellow) and instances (purple).	+	-	-	+	+	Java	OWL, RDF	+
RDF Gravity[‡‡]	2d	Classes and instances as labeled rectangles. Different colors for classes and instances.	+	+	+	+	+	Java	RDF, OWL	+
CropCircles[15]	2d	Classes and instances as concentric circles.	+	-	-	+	+	Java	OWL RDF	+
OWLGrEd[16]	2d	Class as a table (name /properties).	+	+	-	-	+	Java	OWL	+
OntoTrack[17]	2d	Tree nodes are rectangles containing a label.	-	+	-	+	+	Java	OWL	+
Knoocks[20]	2d	Classes represented as blocks (rectangles).Every block stands for a class with its subclasses.	+	+	+	+		C#	OWL Lite	
VOM[§§]	2d	Classes represented as labeled rectangles.	+	+	-	-	+	Java	OWL-DL	+
OntoTrix[21]	2d	Classes represented as labels.	+		+	+	+	Java	OW, RDF	+
"Ontology visualization tools for everyone"										
OWLeasyViz[14]	2d	Classes as labeled ellipses.	+	+	+	+	+		OWL	-
WebVOWL[18]	2d	Classes as labeled circles.	+	+	+	-	+	HTML, CSS JavaScript	OWL	+
ProtégéVOWL[19]	2d	Classes as labeled circles.	+	+	+	-	+	Java	OWL	+

Figura 2.1: Tabella Confronto

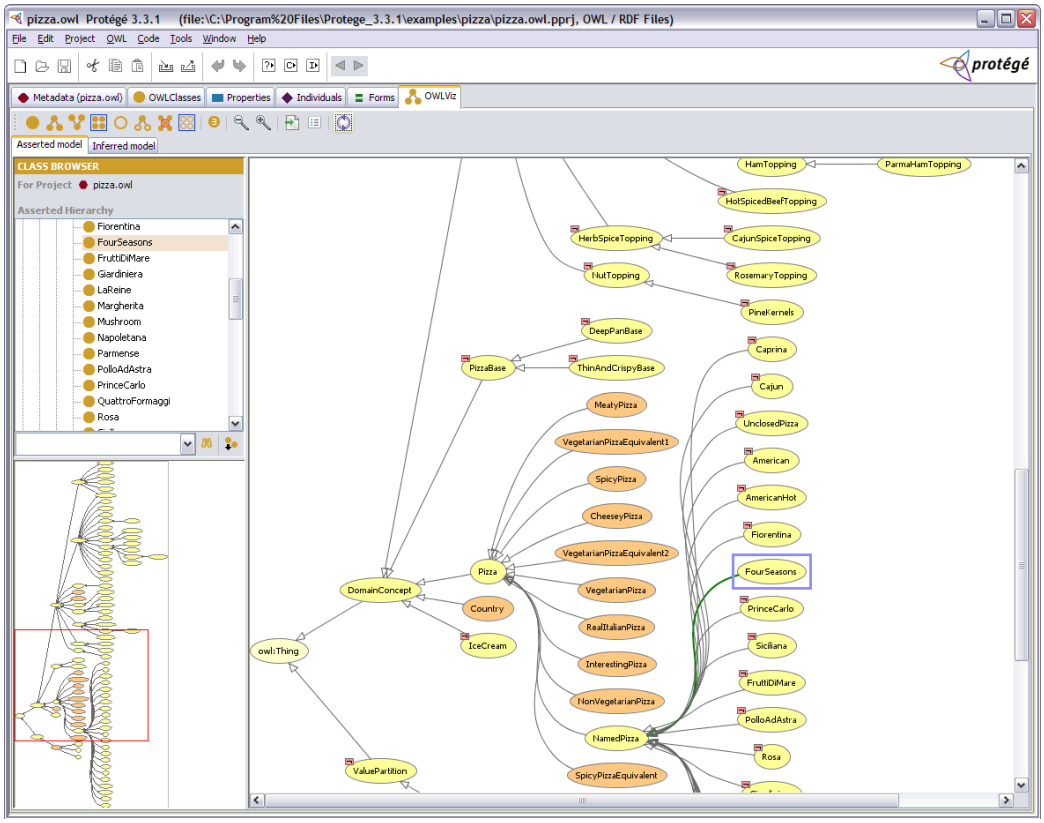

Figura 2.2: Protégé in esecuzione

2.1.2 GrOWL

GrOWL è uno strumento per la modifica e la visualizzazione di ontologie che si basa su OWL-DL (Web Ontology Language- Description Logic). Esso mostra la descrizione logica delle ontologie OWL senza esporne la sintassi complessa. GrOWL mostra automaticamente l'albero dell'ontologia e il relativo grafo. Inoltre le classi di gerarchia di un'ontologia sono illustrate in due forme, un albero di navigazione nella parte sinistra dello schermo e un grafo di classi senza le istanze. Mostra le Tbox e le Abox separatemene. GrOWL è basato sulla libreria Prefuse [2]. Esso è implementato come un applet Java, come plugin per Protégé e come applicazione Java stand alone. La sua politica basata sul design ne assicura l'estensibilità e la possibilità di adattarsi facilmente alla necessità di progetti differenti. Il primo principio che sta alla base di GrOWL è l'uso dei colori, dell'ombreggiatura e delle figure dei nodi per codificare le proprietà del costrutti del linguaggio di base [5] [1].
Opzioni per il layout:

- Zoom
- Filtro, che mostra solo la definizione della classe, della sottoclasse, superclasse o le istanze del nodo selezionato (sia nell'albero che nel grafo è possibile applicare un filtro).
- Ricerca

[1] http://protege.stanford.edu/
[2] http://prefuse.org/

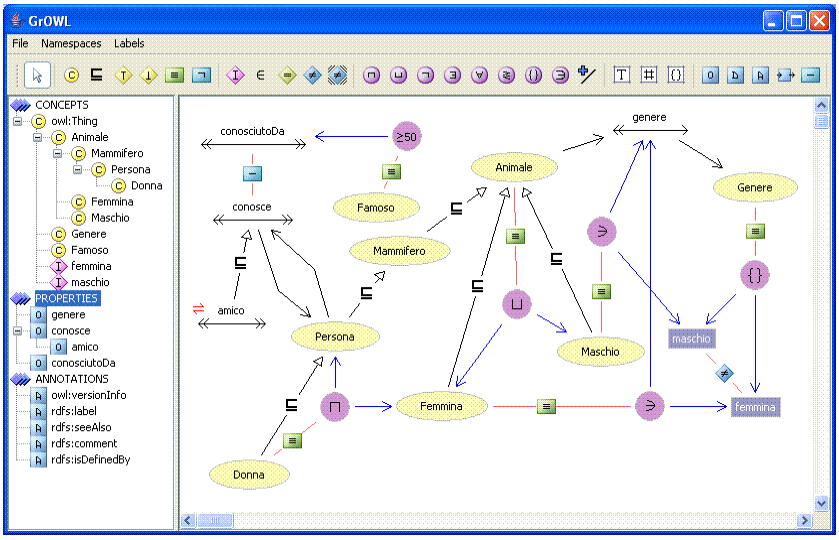

Figura 2.3: GrOWL - visione globale di una semplice ontologia

La sintassi

Figura 2.4: Classe in GrOWL

Le classi sono rappresentate come ellissi etichettate con il nome della classe

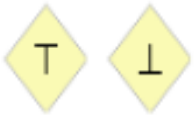

Figura 2.5: owl:Thing e owl:Nothing

owl:Thing e owl:Nothing sono rappresentate da rombi etichettati rispettivamente con i simboli top e bottom.

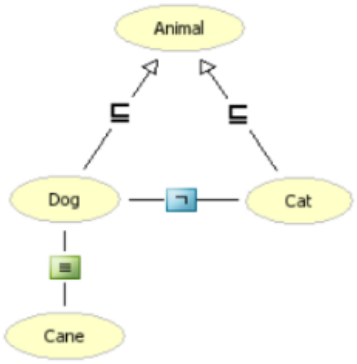

Figura 2.6: Mappatura della tassonomia e dei concetti

Sottoclasse: owl:subClassOf è un arco con una freccia vuota con il simbolo del sottoinsieme.
Equivalenza: due classi equivalenti sono connesse da un arco non direzionale con un simbolo di equivalenza .
Disgiunzione: due classi disgiunte sono connesse da un arco non direzionato con l'etichetta "not".

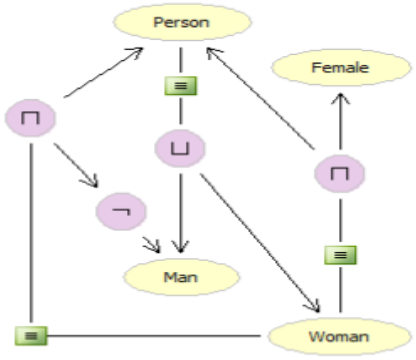

Figura 2.7: Composizione booleana delle classi

Gli *operatori logici* sono rappresentati da cerchi decorati con uno specifico costruttore. Ogni operatore è collegato con il suo argomento da un arco orientato non etichettato.

Figura 2.8: Proprietà

le *Proprietà* sono rappresentate da segmenti orizzontali con delle frecce alle estremità.

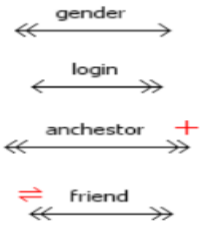

Figura 2.9: Proprietà 2

Le *Proprietà funzionali* hanno una singola freccia sulla destra (Nell'esempio "gender").
Le *Proprietà funzionali inverse* hanno la doppia freccia sulla sinistra ("login" nell'esempio).
Le *Proprietà transitive* hanno un "+" sopra la freccia a destra.
Le *Proprietà simmetriche* hanno una freccia bidirezionale in alto a sinistra (Nell'esempio "friend").

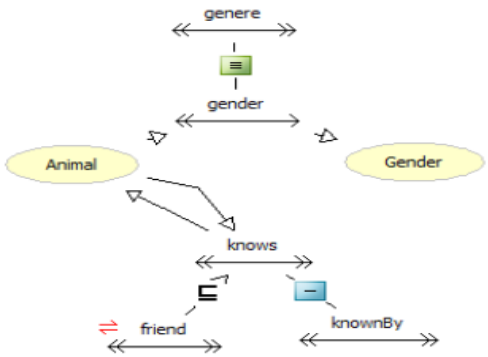

Figura 2.10: Regole degli assiomi

Concetti e *Regole* sono connessi da frecce "vuote" che seguono la direzione Dominio-Proprietà-Range. Nell'esempio la proprietà gender va da Animal a Gender.
Gerarchie ed Equivalenze sono rappresentate nello stesso modo delle gerarchie di classi.

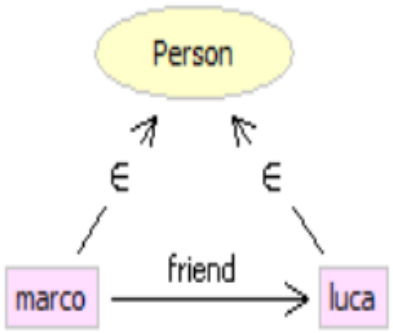

Figura 2.11: Asserzioni nelle ABox

I *Soggetti* sono rappresentati come dei rettangoli insieme all'URI che li identifica. Sono collegati alla classe di cui fanno parte con una freccia etichettata con il simbolo di appartenenza, mentre sono collegati tra di loro da una frec-

cia riportante il nome della regola che li collega. Nell'esempio Marco e Luca sono persone e sono amici.

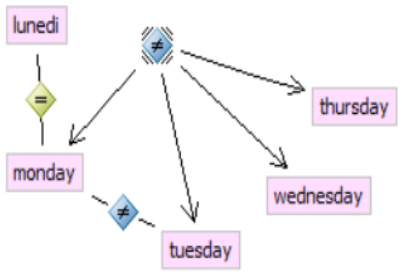

Figura 2.12: Uguaglianza e disuguaglianza tra soggetti

L'uguaglianza e disuguaglianza tra soggetti può essere espressa con degli archi non direzionati (nell'esempio di figura 2.12 Monday è diverso da Tuesdsay) oppure per ridurre il numero di triple è rappresentato in forma "circolare" in cui i soggetti sono collegati con il simbolo"diverso". Nell'esempio (Monday, Tuesday, Wednesday e Thursday sono differenti).

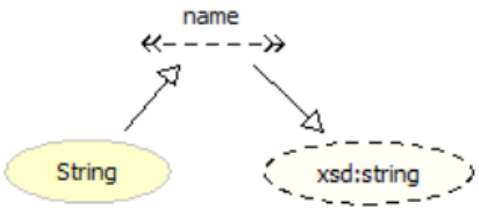

Figura 2.13: Proprietà dei tipi di dato

Le *Proprietà dei tipi di dati* seguono le stesse regole degli oggetti ma sono tratteggiati (nell'esempio "name").
I *Tipi di dato letterali* sono rappresentati con ovali tratteggiati (nell'esempio xsd:string).

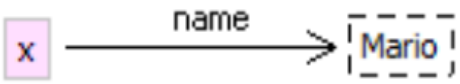

Figura 2.14: Valori letterali

I *Valori letterali* sono inseriti in rettangoli tratteggiati.

Valutazione

GrOWL ha dimostrato una completa capacità di visualizzazione, una discreta usabilità e un alto grado di affidabilità. Un altro merito di GrOWL è quello di essere implementato sia come applicazione stand-alone che come plugin di Protégé. Gli aspetti negativi sono lo scarso grado di personalizzazione del layout e la documentazione [1].

Feature	Voto
Capacità di visualizzazione	5
Usabilità	2
Aspetto estetico	3
Documentazione	2
Personalizzazione	1

Tabella 2.1: Tabella di valutazione di GrOWL

2.1.3 RDF Gravity

RDF GRAph VIusualizazion Tool (RDF Gravity) è uno strumento per visualizzare grafi orientati. Il tool fornisce una semplice ma valida visualizzazione della struttura del grafo oltre che alla possibilità di applicare un filtro per visualizzare solamente specifici frangenti del grafo RDF. RDF Gravity

è sviluppato in Java e richiede la JVM1.3 o superiore. Usa le JUNG graph API[3] e Jena Semantic web toolkit (Jena 2.0). Illustriamo ora le principali funzioni[4]:

Visualizzazione multipla dei nodi

L'utente può selezionare uno o più nodi dal grafo in due modi:

- Selezionando i nodi premendo il tasto sinistro del mouse in modo da formare un area di selezione.

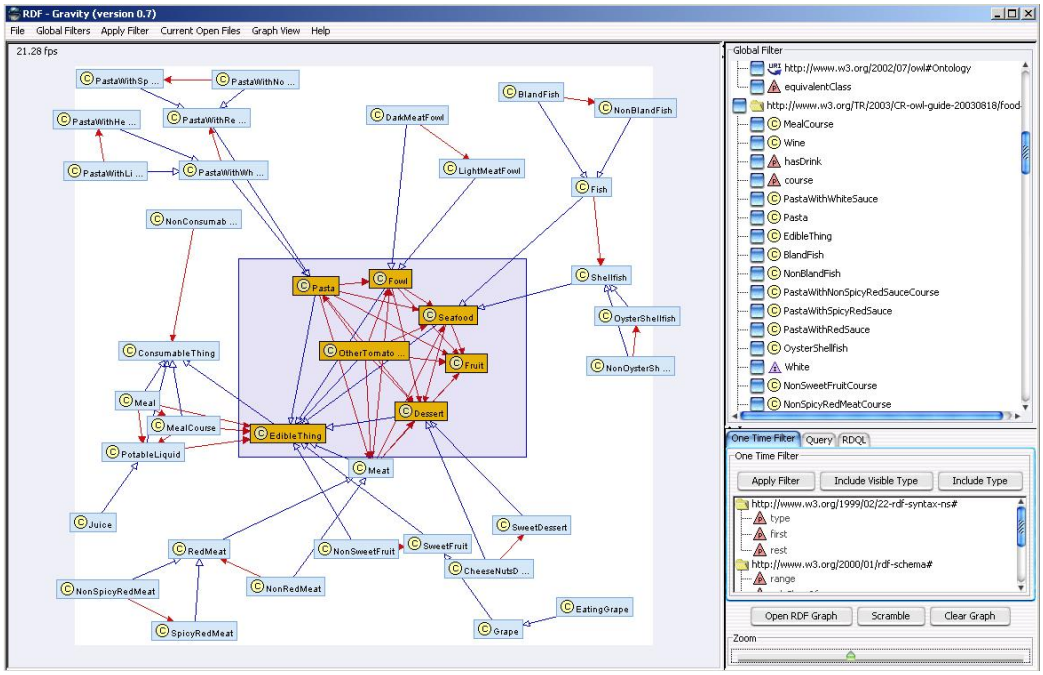

Figura 2.15: Rdf Gravity-selezione 1.

- Premendo il tasto "shift" e selezionando i nodi interessati.

[3]http://jung.sourceforge.net/

[4]immagini prese da http://semweb.salzburgresearch.at/apps/rdf-gravity/

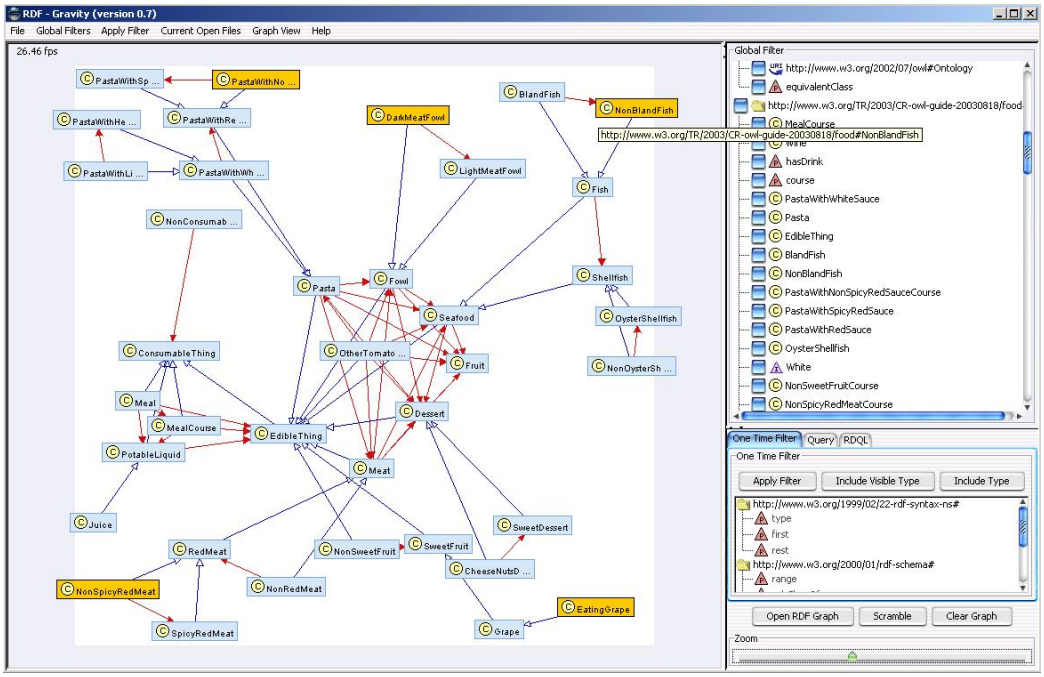

Figura 2.16: Rdf Gravity-selezione 2.

Zoom

RDF Gravity permette all'utente di effettuare uno zoom dell'intero grafo tramite un apposita interfaccia posta in basso a destra della schermata. Durante lo zoom i nodi selezionati rimangono visibili.

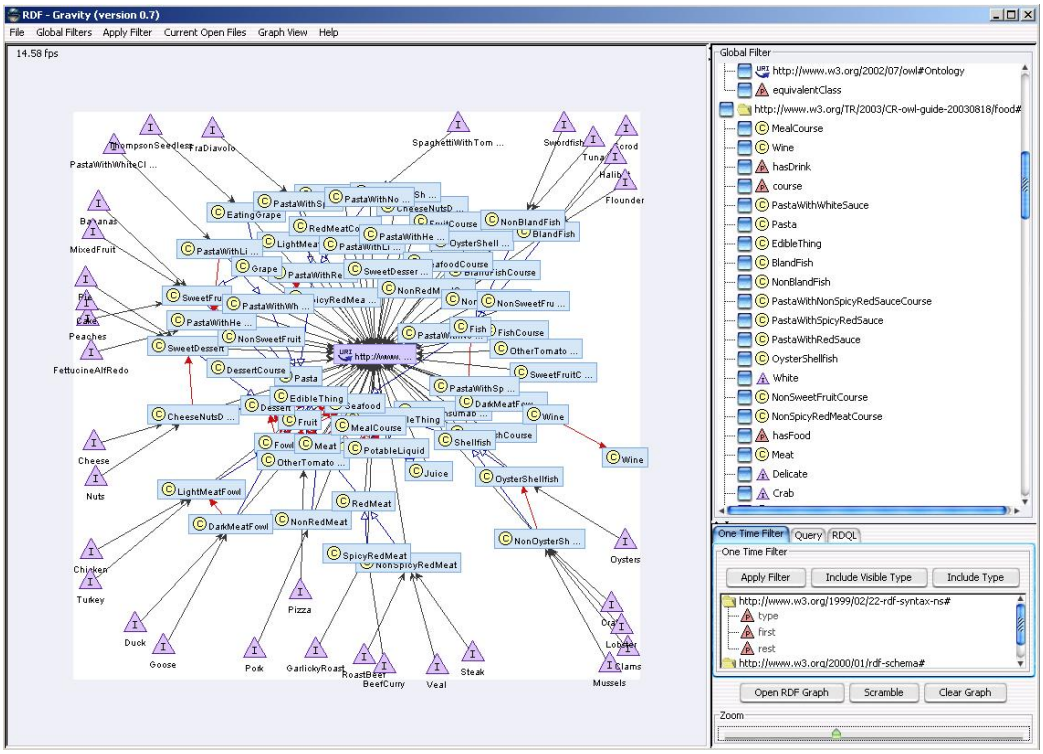

Figura 2.17: Rdf Gravity-zoom 1.

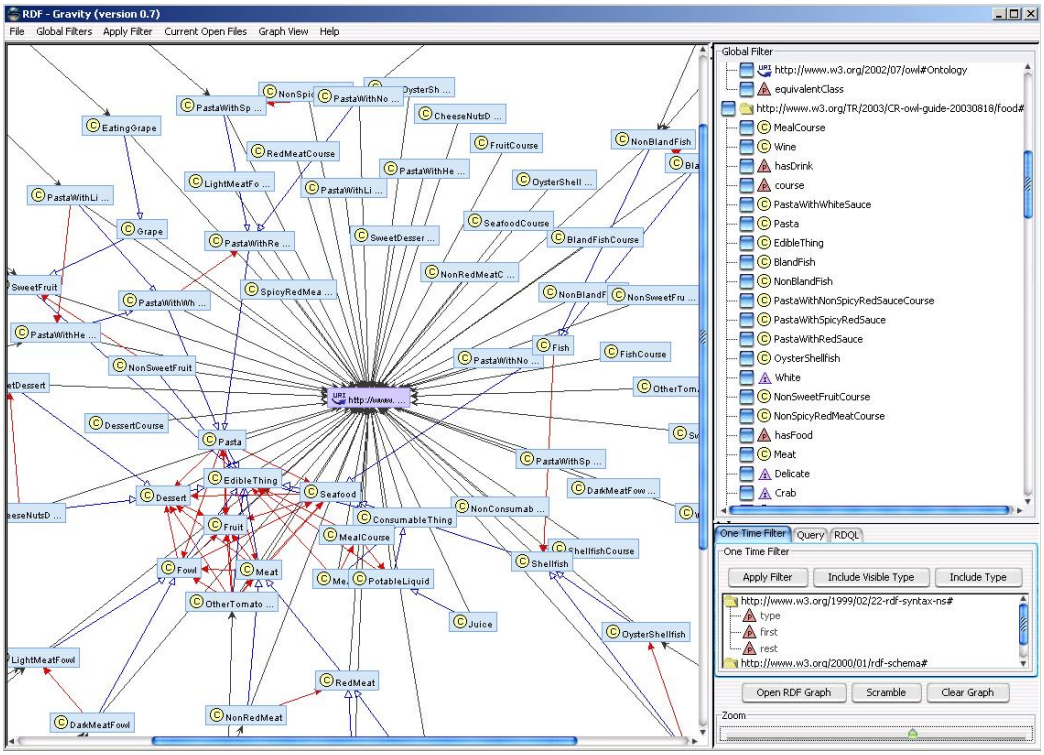

Figura 2.18: Rdf Gravity-zoom 2.

Navigazione dei nodi

Il tool permette di navigare attraverso nodi individuali, mostrando i nodi entranti o uscenti con le relative istanze.

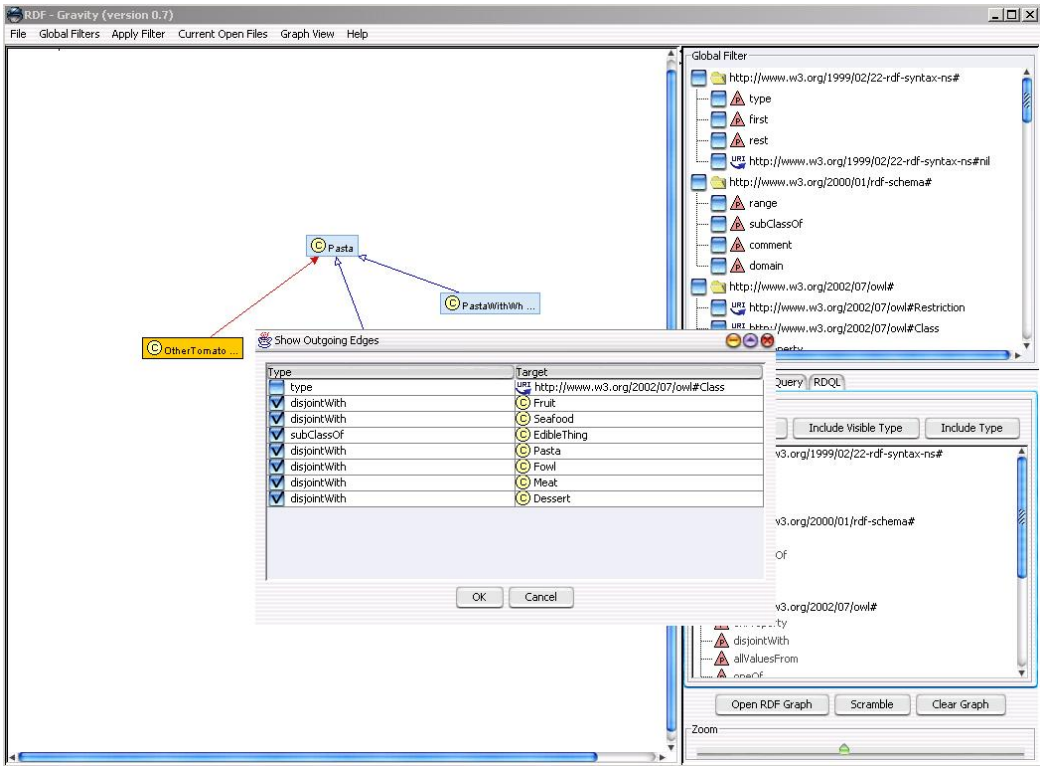

Figura 2.19: Rdf Gravity-navigazione per nodi 1.

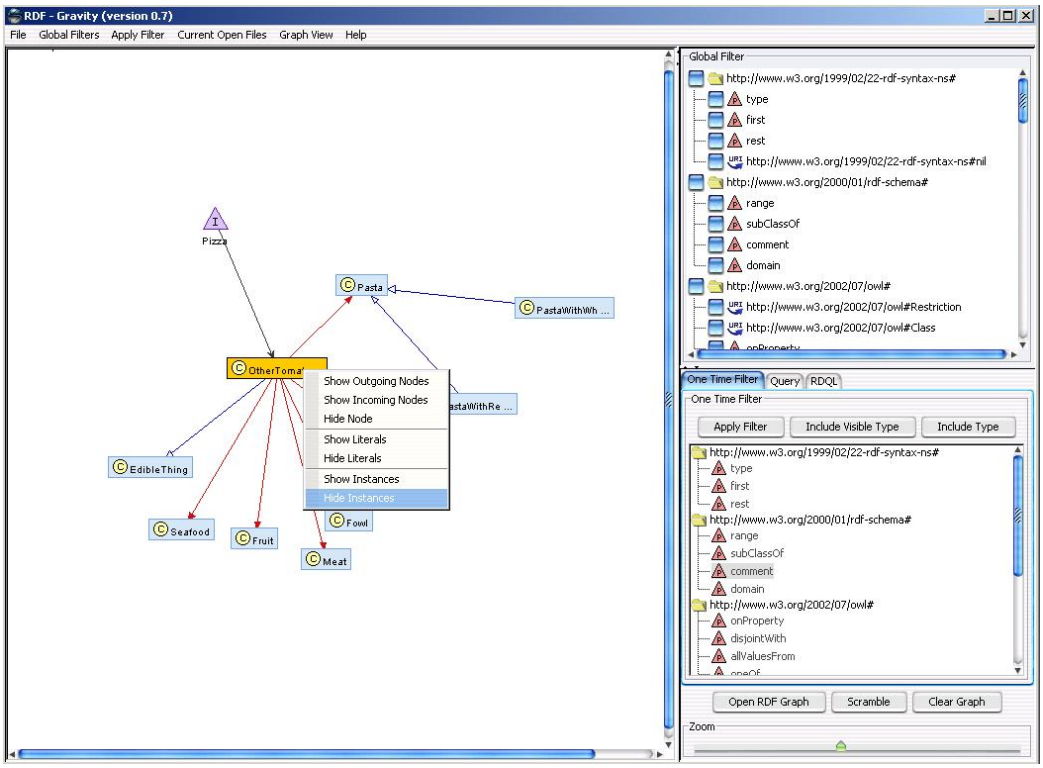

Figura 2.20: Rdf Gravity-navigazione per nodi- instanza.

RDQL Queries

Una caratteristica di notevole importanza presente in questo strumento è la possibilità di effettuare delle query e visualizzare le triple di risposta direttamente nel pannello del grafo. Nella figura sottostante vediamo il risultato della seguente query:

```
SELECT *
  WHERE  (?x, ?y,
  <http://www.w3.org/TR/2003/CR-owl-guide-20030818/food#Pasta>)
```

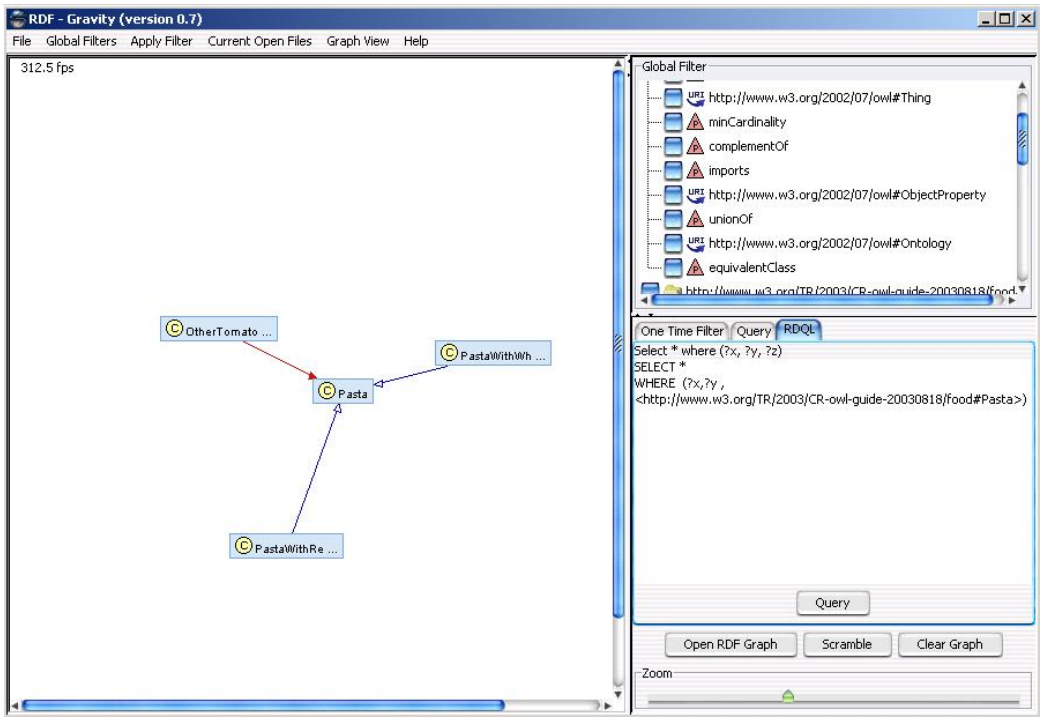

Figura 2.21: Rdf Gravity-RDQL Query

Valutazione

RDF Gravity è uno strumento molto utile e interessante, sicuramente uno dei più completi per la categoria "Strumenti di visualizzazione di ontologie per esperti". Facendo parte però di questa categoria si presta male all'utilizzo da parte degli utenti non esperti, infatti il tool richiede una buona conoscenza e affinità nell'ambito delle ontologie e delle query.

Feature	Voto
Capacità di visualizzazione	5
Usabilità	3
Aspetto estetico	2
Documentazione	3
Personalizzazione	1

Tabella 2.2: Tabella di valutazione di RDF Gravity

2.2 Strumenti di visualizzazione per tutti

2.2.1 DBpedia

La città di Bologna è identificata sul dataset di DBpedia.org nel seguente modo
http://dbpedia.org/resource/Bologna
Inserendo questo link nel browser saremo reindirizzati alla seguente pagina:
http://dbpedia.org/page/Bologna
Da questa pagina è possibile visualizzare tutte le proprietà collegate alla risorsa come per esempio l'area territoriale occupata da Bologna che ammonta a 140.7 km quadrati, possiamo ricavare il prefisso telefonico, il sindaco, il codice postale e tutta una serie di informazioni relative alla nostra risorsa. Le figure 2.22 e 2.23 mostrano i predicati e gli oggetti associati alla città di Bologna. Quest'ultimi possono essere dei semplici dati finiti come il codi-

ce postale "40100" oppure dei collegamenti ipertestuali che possono essere esplorati a loro volta come il sindaco Merola (dbr:Virginio_Merola).

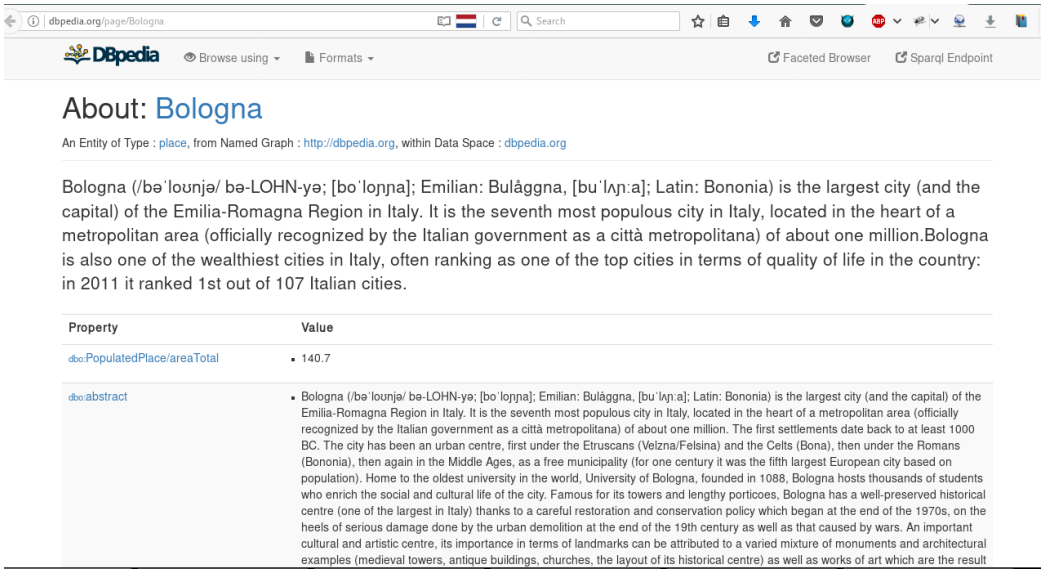

Figura 2.22: http://dbpedia.org/page/Bologna -pagina 1.

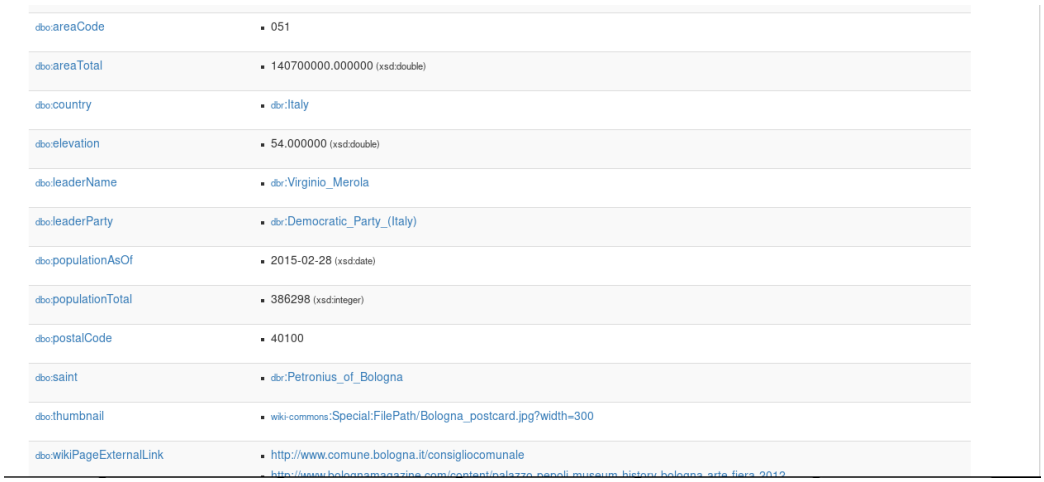

Figura 2.23: http://dbpedia.org/page/Bologna -pagina 2.

Valutazione

Questo tipo di navigazione è prettamente ipertestuale in cui si può esplorare solo in un unica direzione cioè dalla risorsa (Bologna) verso i predicati e oggetti associati, ma non si può vedere, per esempio, a quali altre risorse Bologna è associata come oggetto. Non è fornita nessuna documentazione. Inoltre la rappresentazione che ci viene fornita dal tool è analitica e non fornisce una panoramica generale come può fare un grafo orientato.

Feature	Voto
Capacità di visualizzazione	2
Usabilità	5
Aspetto estetico	3
Documentazione	s.v
Personalizzazione	1

Tabella 2.3: Tabella di valutazione di DBpedia.

2.2.2 Memo Graph

Memo Graph è pensato per essere usato da tutti, sia dagli esperti, sia da chi non ha familiarità con le ontologie. Memo Graph rappresenta l'ontologia come un grafo. L'algoritmo di visualizzazione presenta 3 vantaggi:

- Ottimizzazione dello spazio del monitor.
- Aiuta a riflettere sull'importanza delle classi nel grafo disegnato: posiziona i nodi in modo che le classi con più connessioni risultino al centro della visualizzazione e quelle meno connesse all'estremità.
- Aumenta la leggibilità del grafo facendo in modo che gli archi che connettono i nodi non si sovrappongano o si intralcino.

Memo Graph visualizza tutti gli elementi chiave di un'ontologia come classi, istanze, proprietà dei tipi di dato e proprietà degli oggetti. Le regole di

relazione tra i nodi sono espresse con archi etichettati. Al contrario di molti altri progetti, MemoGraph identifica i nodi con delle immagini o con delle etichette. Questa tecnica facilita la comprensione e rende i nodi distinti gli uni dagli altri. Le immagini sono estratte direttamente da Google se non vengono fornite dall'utente.

Memo Graph supporta la ricerca in due modi: Tramite una parola chiave ed evidenziando uno specifico elemento nel grafo. Nel primo caso abbiamo a disposizione due modi per digitare: manualmente o tramite la scrittura vocale. Per quanto riguarda il secondo metodo, quando selezioniamo un nodo i suoi dettagli vengono mostrati in basso nella schermata. La sua interfaccia si suddivide in tre sezioni:

- "MEMO GRAPH viewer": visualizza l'ontologia sotto forma di grafo.
- "MEMO GRAPH details": elenca le informazioni riguardanti il nodo selezionato.
- "MEMO GRAPH search": fornisce le opzioni di ricerca per la parola chiave.

Applicazioni di Memo Graph

Memo Graph è integrato in un progetto in via di sviluppo chiamato CAPITAIN MEMO. Questo è un progetto rivolto agli anziani che propone un ontologia chiamata PersonLink volta a modificare, memorizzare e ragionare sui collegamenti delle relazioni familiari. PersonLink definisce rigorosamente le relazioni che intercorrono tra i membri di una famiglia tenendo conto delle differenze esistenti tra le culture e i linguaggi, includendo nuovi tipi di relazioni che emergono dalla società dei giorni nostri. In particolare CAPTAIN MEMO è un applicazione stand-alone rivolta ad aiutare persone che presentano i primi sintomi dell'Alzheimer, aiutandoli a navigare nel proprio albero genealogico. CAPTAIN MEMO è basato su ontologie RDF che descrivono persone, animali e relazioni esistenti tra questi [3].

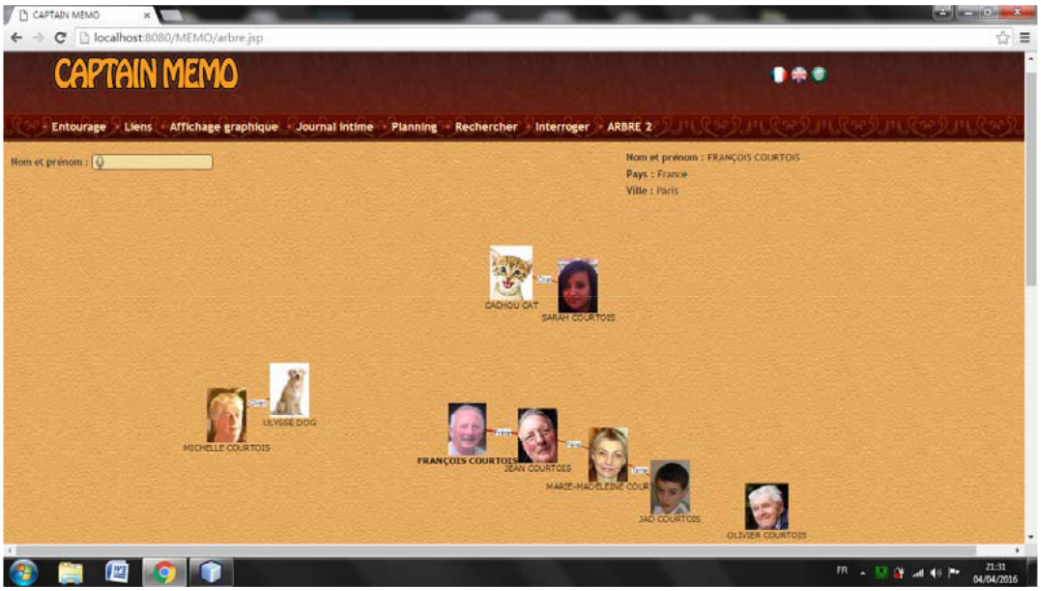

Figura 2.24: Captain Memo su windows.

Valutazioni

Memo Graph si è rivelato uno strumento molto utile sopratutto in relazione alla sua applicazione nel progetto CAPTAIN MEMO. Dimostra di essere uno strumento molto efficace in particolare se utilizzato da persone non esperte di ontologie. Fornisce dei componenti interessanti come la ricerca tramite parola chiave e il reperimento di immagini automatizzato. Il limite principale è il suo scarso grado di personalizzazione che lo rendo uno software troppo statico e, essendo uno strumento progettato nel 2016 e ancora in via di sviluppo non è ancora reperibile nessuna documentazione.

2.2.3 WebVOWL

VOWL è un tool molto utile per la visualizzazione di ontologie. Esso è disponibile in due forme: come plugin di Protégé, chiamato appunto ProtégéVOWL, e in versione web app sotto il nome di WebVOWL. In questa sezione andremo a illustrare la versione web. WebVOWL sfrutta le librerie Prefuse ed è scritto in JavaScript. L'ontologia , per essere disegnata,deve

Feature	Voto
Capacità di visualizzazione	2
Usabilità	5
Aspetto estetico	3
Documentazione	s.v
Personalizzazione	2

Tabella 2.4: Tabella di valutazione di MEMOgraph.

essere convertita in un file JSON, il quale permetterà la creazione automatica della visualizzazione. Esiste un convertitore chiamato OWL2VOWL implementato in Java e basato sulle OWL API che è usato per adempire questo compito. L'ontologia così si rende disponibile sotto forma di force-directed graph in accordo con le specifiche di VOWL. Le tecniche di iterazione di VOWL consentono una facile esplorazione e una personalizzazione dell'esperienza.

Preprocesso delle ontolgie

Al posto di essere legati ad un particolare parser OWL, viene definito un JSON schema per WebVOWL in cui le ontologie devono essere convertite. Il formato differisce dalle tipiche serializzazioni in OWL per permettere l'efficiente generazione del grafo interattivo. Per questa ragione è differente anche da altri JSON schema presenti nel contesto del Semantic Web come RDF/JSON e JSON-LD. Il file VOWL-JSON contiene la TBox dell'ontologia come classi, proprietà e tipi di dato insieme al tipo di informazione *(owl:ObjectProperty, xsd:dateTime, etc.)*, inoltre è possibile anche eseguire delle query ad un dataset tramite un interfaccia grafica interattiva che illustreremo nel capitolo successivo. Le caratteristiche addizionali degli elementi (per esempio elementi funzionali o deprecati), le informazioni di intestazione (titolo dell'ontologia, versione, ecc) e le statistiche opzionali dell'ontologia (numero di classi,proprietà ecc) sono elencate separatamente. Se queste ulti-

me informazioni non sono fornite nel file JSON saranno calcolate a run-time da WebVOWL. Nonostante WebVOWL è basato su JavaScript, la trasformazione dell'ontologia OWL nel formato JSON non necessita di essere eseguita con JavaScript ma può essere fatta con altri linguaggi di programmazione. Il convertitore OWL2VOWL è basato su Java e usa le già collaudate OWL API dell'Università di Manchester. Il convertitore accede alla rappresentazione fornita dalle OWL API e trasforma nel formato JSON richiesto da WebVOWL.

Visualizzazione con WebVOWL

La visualizzazione avviene a run-time tramite i dati forniti dal file JSON. WebVOWL rende gli elementi grafici in accordo con le specifiche di VOWL, per esempio prendendo lo stile dal codice SVG e CSS fornite nelle specifiche. Il grafo è creato con una libreria JavaScript chiamata D3.js (Data Driven Document). Questa simula una forza fisica da applicare ai nodi ed il risultato è un grafico animato in cui la posizione dei nodi è dinamica. Possiamo vedere un esempio nella a sottostante.

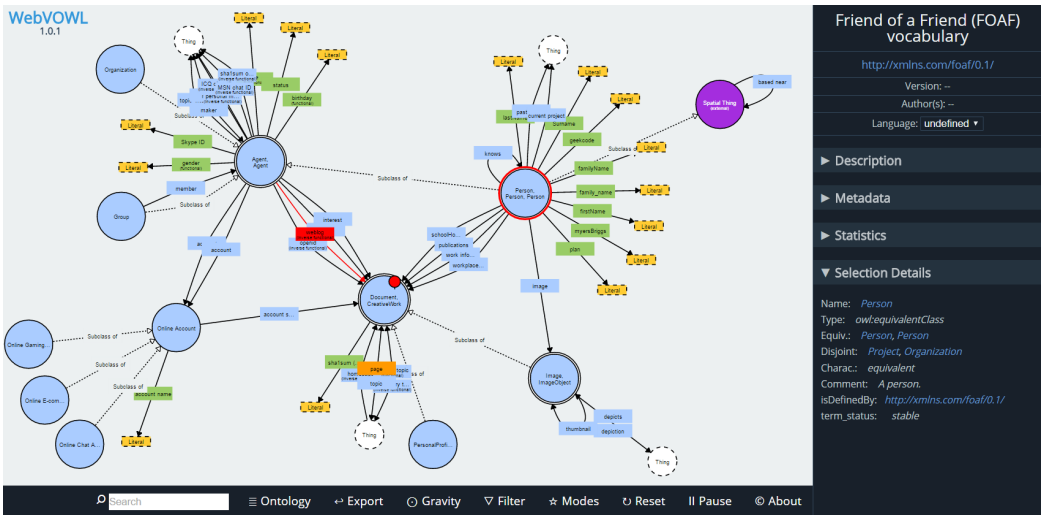

Figura 2.25: WebVOWL in esecuzione

Interazione e esplorazione

L'utente può ottimizzare la visualizzazione del grafo e adattarlo alle sue esigenze posizionando i nodi semplicemente trascinandoli. La disposizione del grafo può anche essere sistemata modificando le impostazioni della forza fisica tra i nodi e della gravità. I tipi di dato hanno una forza separata così possono essere posizionati vicini alla classe alla quale sono collegati. Ogni volta che i nodi vengono trascinati l'algoritmo viene innescato e il resto dei nodi vengono riposizionati con una transizione animata. Per prevenire questo comportamento l'utente può mettere in pausa la disposizione automatica in favore di un posizionamento manuale dei nodi. L'utente può inoltre effettuare uno zoom e selezionare gli elementi del grafico. I dettagli sui nodi selezionati sono elencati nell'apposito widget sulla sidebar. Un altra caratterista è il filtro del dati, che permette di diminuire la quantità di nodi presenti nel grafo. Infine WebVOWL consente di esportare l'intero grafo come immagine SVG che può essere aperta in altre applicazioni, scalata senza perdita di qualità, modificata, condivisa e stampata [6].

Valutazione

WebVOWL è sicuramente lo strumento più interessante presentato fin'ora in questo documento. Le sue caratteristiche permettono l'utilizzo anche a coloro che sono meno affini alle ontologie. La sua implementazione come web-app rende ancora più accessibile questo tool. Inoltre l'interfaccia risulta molto accattivante e possiede le giuste caratteristiche per garantire un esperienza di esplorazione gratificante all'utente. Purtroppo però non è ancora possibile personalizzare la rappresentazione degli elementi del grafo.

Feature	Voto
Capacità di visualizzazione	5
Usabilità	4
Aspetto estetico	4
Documentazione	4
Personalizzazione	1

Tabella 2.5: Tabella di valutazione di WebVOLW

Capitolo 3

JLO: un formato per visualizzare dati Semantici

In questo capitolo descriveremo il progetto di una web app basata su JLO per la visualizzazione di dati Semantici rivolto ad utenti non esperti e illustreremo le 4 fasi che portano al suo completo sviluppo. In particolare questa applicazione consentirebbe all'utente di eseguire delle query ad un dataset specifico e di personalizzare la visualizzazione delle risposte tramite un'interfaccia grafica interattiva, permettendo quindi un semplice utilizzo a chiunque. Il progetto si sviluppa in 4 fasi:

- Preparazione ed esecuzione della query al dataset.
- Personalizzazione del layout tramite un interfaccia grafica.
- Creazione del file JLO contenente i dati necessari.
- Esecuzione di uno script che crei il grafo prendendo in input il file JLO.

In questo documento mi sono occupato di implementare gli ultimi due punti, lasciando i primi due come sviluppi futuri. Descriveremo ora i quattro punti nel dettaglio ipotizzando di voler interrogare il dataset di DBpedia.

3.1 Preparazione ed esecuzione della query

Ipotizziamo di voler chiedere a DBpedia in quali squadre hanno giocato tre giocatori di calcio come Gianluigi Buffon, Paul Pogba e Cristiano Ronaldo, e ottenere come risultato delle triple in cui vediamo il nome del soggetto, il predicato e il nome dell'oggetto. Una query plausibile potrebbe essere la seguente:

```
SELECT DISTINCT ?s ?p ?o WHERE{
                      {?sogg a dbo:Person;
                            rdfs:label ?s;
                            rdfs:label "Cristiano Ronaldo"@en;
                            ?pred ?oo.
                       ?pred rdfs:label ?p;
                              rdfs:label "team"@en.
                           ?oo rdfs:label ?o.
                      FILTER (lang(?s) = 'en')
                      FILTER (lang(?p) = 'en')
                      FILTER (lang(?o) = 'en')}UNION
{
                      ?sogg a dbo:Person;
                            rdfs:label ?s;
                            rdfs:label "Gianluigi Buffon"@en;
                            ?pred ?oo.
                       ?pred rdfs:label ?p;
                          rdfs:label "team"@en.
                       ?oo rdfs:label ?o.
FILTER (lang(?s) = 'en')
FILTER (lang(?p) = 'en')
FILTER (lang(?o) = 'en')
}
```

```
UNION


      {
                        ?sogg a dbo:Person;
                              rdfs:label ?s;
                              rdfs:label "Paul Pogba"@en;
                              ?pred ?oo.
                         ?pred rdfs:label ?p;
                            rdfs:label "team"@en.
                         ?oo rdfs:label ?o.
FILTER (lang(?s) = 'en')
FILTER (lang(?p) = 'en')
FILTER (lang(?o) = 'en')
}
     }
```

Il risultato di questa richiesta è mostrato in tabella 3.1

Per un utente che non ha dimestichezza con le query e le ontologie, estrapolare un risultato del genere è impensabile. Per questo motivo credo che un'interfaccia simile a quella implementata da VOWL sia una scelta plausibile. L'utente può fare delle ricerche e scegliere una serie di parametri tra quelli che gli vengono proposti mentre l'applicazione esegue la query senza mostrarne il codice complesso. Nelle figure 3.2, 3.3 e 3.4 vengono illustrati i passi necessari per effettuare una richiesta a DBpedia. Ipotizziamo di volere richiedere le squadre in cui ha giocato Gianluigi Buffon. Per prima cosa cerchiamo il nostro soggetto nella barra di ricerca selezionandolo tra quelli che ci vengono proposti. Nella schermata centrale verrà disegnato un nodo a forma di cerchio che lo rappresenta. Successivamente cerchiamo il predicato nella barra di ricerca, in questo caso è "dbo:team". Verrà creato

s	p	o
"Cristiano Ronaldo"@en	"team"@en	"Manchester United F.C."@en
"Cristiano Ronaldo"@en	"team"@en	"Real Madrid C.F."@en
"Cristiano Ronaldo"@en	"team"@en	"C.D. Nacional"@en
"Cristiano Ronaldo"@en	"team"@en	"Portugal national under-20 football team"@en
"Cristiano Ronaldo"@en	"team"@en	"Portugal national under-17 football team"@en
"Cristiano Ronaldo"@en	"team"@en	"CF Andorinha"@en
"Cristiano Ronaldo"@en	"team"@en	"Sporting Clube de Portugal"@en
"Gianluigi Buffon"@en	"team"@en	"Italy national under-21 football team"@en
"Gianluigi Buffon"@en	"team"@en	"S.S.D. Parma Calcio 1913"@en
"Gianluigi Buffon"@en	"team"@en	"Italy national football team"@en
"Gianluigi Buffon"@en	"team"@en	"Juventus F.C."@en
"Paul Pogba"@en	"team"@en	"Juventus F.C."@en
"Paul Pogba"@en	"team"@en	"Manchester United F.C."@en
"Paul Pogba"@en	"team"@en	"France national under-17 football team"@en
"Paul Pogba"@en	"team"@en	"Le Havre AC"@en
"Paul Pogba"@en	"team"@en	"France national under-16 football team"@en
"Paul Pogba"@en	"team"@en	"France national under-18 football team"@en

Figura 3.1: Risultato query 1

un rettangolo accanto al nodo a forma di cerchio nel riquadro centrale. Per ultima cosa selezioniamo il cerchio vuoto in alto a destra che rappresenterà i risultati della query e trasciniamolo nel riquadro centrale, unendo le figure con dei collegamenti che ci vengono forniti cliccando sui nodi.

3.2 Personalizzazione del Layout

Questa fase è molto importante perchè è da questa che dipende la generazione del JLO, il cuore del progetto. Abbiamo presupposto che l'utente abbia eseguito una query ad un dataset, vediamo ora come può personalizzare la visualizzazione del risultato.

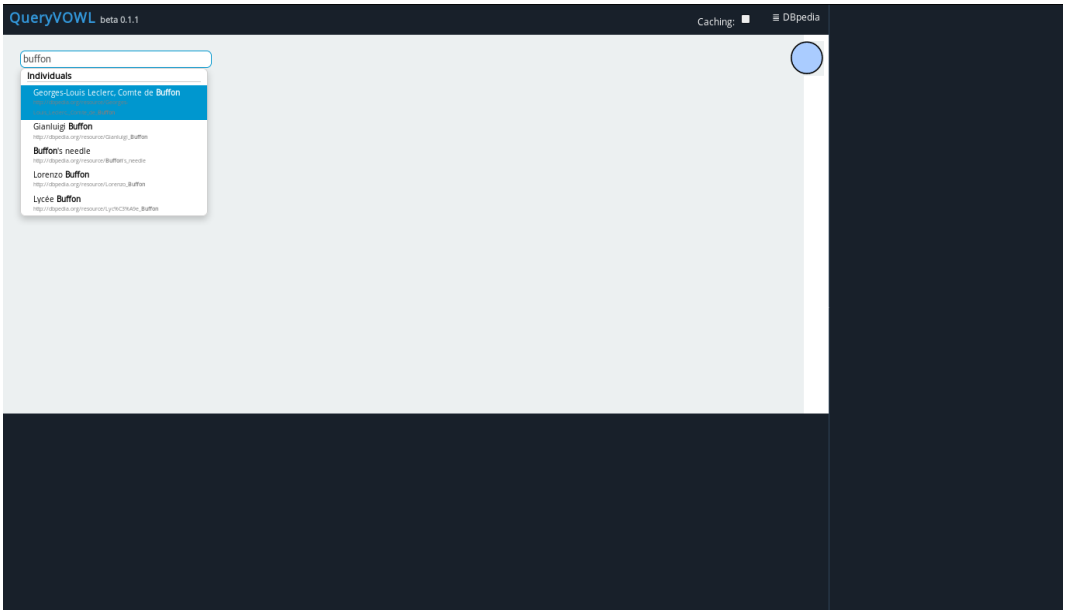

Figura 3.2: L'interfaccia grafica per eseguire query di vowl. Selezioniamo il soggetto Gianluigi Buffon cercandolo dalla barra di ricerca.

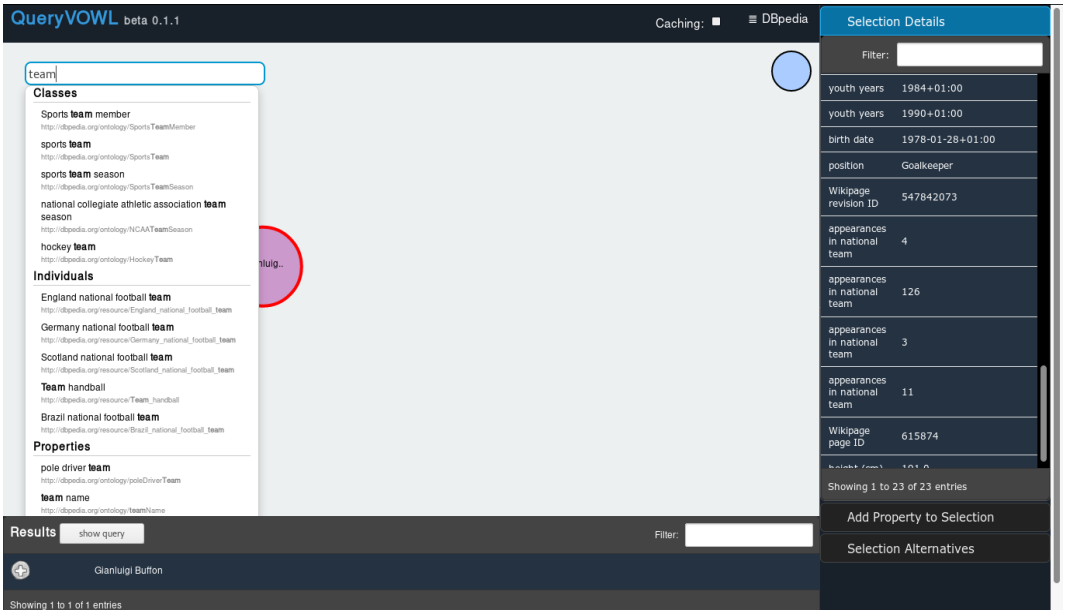

Figura 3.3: L'interfaccia grafica per eseguire query di vowl. Selezioniamo il predicato dbo:team ricercandolo nella barra di ricerca.

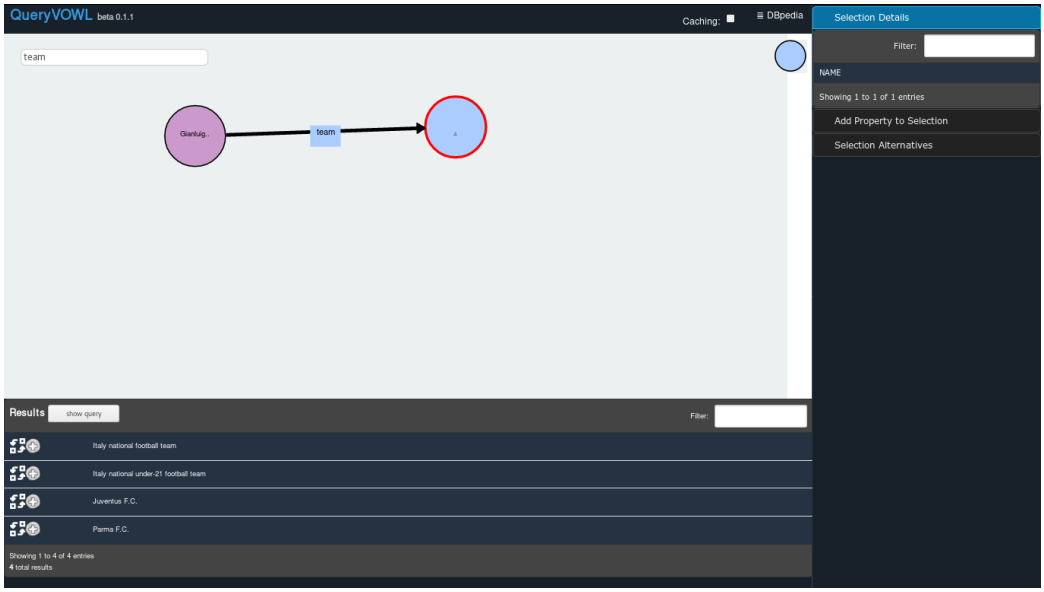

Figura 3.4: L'interfaccia grafica per eseguire query di vowl. Selezioniamo un nodo vuoto e colleghiamolo al soggetto "Buffon" per vedere i risultati della query nella sezione in basso.

La scelta delle forme

Il grafo che andremo a costruire è composto da nodi e archi. I primi si dividono fondamentalmente in due categorie: I soggetti e gli oggetti delle triple che visualizzeremo come risposta alla query. Gli archi individuano i collegamenti tra questi mettendo in evidenza il predicato. Le specifiche di JLO consentono di personalizzare la visualizzazione di ogni singolo nodo scegliendo tra 6 tipi di template:

- Cerchio
- Rettangolo
- Ellisse
- Esagono
- Immagine (JPG o PNG)

- Un template personalizzato

Per i primi quattro è obbligatorio fornire una dimensione e un colore , sarà poi compito dello script, che prenderà in input il file JLO, scalare la dimensione fornita dall'utente in quattro grandezze standard.
Oltre alla scelta della rappresentazione del nodo è possibile personalizzare il bordo delle figure geometriche (tranne che per l'immagine quindi) scegliendo in particolare:

- Lo spessore
- Il colore
- Il tratteggiamento del bordo

Lo standard JLO permette di decidere un azione che verrà eseguita quando la freccia del mouse è posizionata sopra il nodo per evidenziarne il selezionamento. Tra queste troviamo:

- L'aumento dello spessore del bordo
- Il cambio di colore

Per consentire una rappresentazione uniforme dei dati, il formato permette di creare un template personalizzato che può essere assegnato a più nodi. Questo comporta un risparmio di tempo per l'utente e un omogeneità nella visualizzazione di nodi concettualmente simili. Il template personalizzato fornisce, di base, il tipo di nodo (ellisse, cerchio, rettangolo, esagono o immagine) e la grandezza ma è concesso anche definire i colori e le proprietà del bordo. Inoltre è possibile personalizzare i collegamenti tra i nodi, indicando il colore, il tratteggiamento e la direzione.
Per definire le proprietà stilistiche tra tipi di nodi differenti è concessa la creazione di classi personalizzate. Queste oltre che ai colori e alle proprietà dei bordi possono assegnare le azioni da eseguire al passaggio del mouse sul nodo.

3.3 Creazione del file JLO

JLO si basa su JSON (JavaScrip Object Notation). JSON è un semplice formato per lo scambio di dati. Per le persone è facile da leggere e scrivere, mentre per le macchine risulta facile da generare e analizzarne la sintassi (http://www.json.org/json-it.html). JLO è fondamentalmente un JSON composto da quattro array che descrivono rispettivamente i nodi, i link, i template (opzionale) e le classi (opzionale) che contengono caratteristiche comuni ai nodi.

JSON schema

JSON Schema specifica un formato basato su JSON per definirne la struttura dei dati al fine di una validazione. JSON Schema fornisce una sorta di contratto tra i dati JSON e l'applicazione che li richiede e definisce come questi posso variare. JLO viene convalidato e descritto dettagliatamente tramite un JSON Schema reperibile al seguente indirizzo
https://github.com/LucaBonini/JLO/blob/master/README.md

I Nodi

Ogni nodo nell'array dei nodi ha i seguenti campi:

- Id: è una stringa contenente l'URI della risorsa
- Label: è una stringa contente l'etichetta da visualizzare nel nodo.
- Size: è un intero che fornisce la dimenzione del nodo.
- Popup_mo: Popup mouseover è di tipo stringa. Le stringhe accettate sono "resource", "label", o "comment" se si vuole mostrare nel popup l'URI della risorsa, l'etichetta o un commento.

- Mouseover: è una stringa e può essere “border” o “fill” se si vuole far raddoppiare lo spessore del bordo o far cambiare colore del nodo al passaggio del mouse.
- Color_mo: è una stringa che identifica il colore del nodo al passaggio del mouse.
- Cssclass: è una stringa che identifica la classe di appartenenza.
- Comment: è una stringa contente il commento da mostrare come popup.
- Shape: è una stringa che definisce il tipo di nodo. Le stringhe accettate sono “circle”, “rect”, “polygon”, “ellipse”, “image” oppure il nome di un template definito nell’array ”templates”.
- Sstyle: è un array che contiene le proprietà di stile del nodo.
  - Fill: è una stringa che definisce il colore del nodo.
  - Stroke: è una stringa che definisce il colore del bordo.
  - Stroke_width: è un intero che definisce lo spessore del bordo del nodo.
  - Stroke_dasharray: è un intero che definisce il tratteggiamento del bordo. “0” significa non tratteggiato.
  - Text_color: è una stringa che definisce il colore dell’etichetta del nodo.

L’attributo “cssclass” e “sstyle” non sono obbligatori ma almeno uno dei due deve essere presente. Segue un esempio:

```
"nodes": [{
   "id": "http://dbpedia.org/resource/Cristiano_Ronaldo" ,
   "label": "Cristiano Ronaldo",
   "size": 21,
```

```
   "popup_mo":"resource",
   "mouseover":"border",
   "color_mo":"aquamarine",
   "cssclass":"",
   "shape":"template1",
   "sstyle":[{
      "stroke":"black",
      "stroke_width":2,
      "stroke_dasharray":0,
      "text_color":"black"}
   ]
},
...
]
```

I Link

Ogni link ha i seguenti campi:

- Source: è una stringa che contiene l'URI del nodo sorgente.
- Target: è una stringa che contiene l'URI del nodo destinazione.
- Sstyle: è un array che contiene le proprietà stilistiche del link.
  - Direction: è una stringa che identifica la direzione del link. Le stringhe accettate sono "forward", ""back" o "both". Quest'ultimo non è ancora implementato.
  - Fill: è una stringa che identifica il colore del link.
  - Stroke_dasharray: è un intero che definisce il tratteggiamento del link. "0" significa non tratteggiato
  - Stroke_width: è un intero che definisce lo spessore del link.

- Label: è un array che contiene le informazioni relative all'etichetta del predicato posta al centro del link.
  - Id: è una stringa che contiene la label del predicato.
  - Fill: è una stringa che definisce il colore dell'etichetta.
  - Stroke: è una stringa che definisce il colore del bordo dell'etichetta.
  - Stroke_dasharray: è un intero che definisce il tratteggiamento del bordo dell'etichetta.

Anche in questo caso i campi "cssstyle" e "sstyle" non sono obbligatori me deve essere presente almeno uno dei due.

```
"links":[{
   "source": "http://dbpedia.org/resource/Manchester_United_F.C." ,
   "target": "http://dbpedia.org/resource/Cristiano_Ronaldo",
   "cssclass": "",
   "sstyle":[{
        "fill":"black",
        "stroke_dasharray":0,
        "stroke_width":4,
         "direction":"forward"}
  ],
"label":[{
     "id":"team",
      "fill":"blue",
      "stroke":"black",
      "stroke_dasharray":0}
  ]
},
...
]
```

Templates

L'array dei template ha degli elementi contenenti ognuno i seguenti campi:

- Name: è una stringa che identifica il template.
- Shape: è una stringa che definisce il tipo di nodo. Le stringhe accettate sono "circle", "rect", "polygon", "ellipse", "image".
- Fill: è una stringa che definisce il colore del nodo.
- Size: è un intero che definisce la grandezza del nodo.
- Stroke: è una stringa che definisce il colore del bordo.
- Stroke_width: è un intero che definisce lo spessore del bordo del nodo.
- Stroke_dasharray: è un intero che definisce il tratteggiamento del bordo. "0" significa non tratteggiato.
- Text_color: è una stringa che definisce il colore.

```
"templates":[{
   "name":"template1",
   "shape":"ellipse",
   "fill":"red",
   "stroke":"black",
   "size":80,
   "stroke_width":10,
   "stroke_dasharray":3,
   "text_color":"black"
},
...
]
```

Classi

L'array delle classi contiene elementi con i seguenti campi:

- Classname: è una stringa che identifica la classe.
- Fill: è una stringa che definisce il colore del nodo.
- Stroke: è una stringa che definisce il colore del bordo.
- Stroke_width: è un intero che definisce lo spessore del bordo del nodo.
- Stroke_dasharray: è un intero che definisce il tratteggiamento del bordo. "0" significa non tratteggiato.
- Text_color: è una stringa che definisce il colore dell'etichetta del nodo.

```
"css_classes":[{
   "classname":"team",
   "fill":"gold",
   "stroke":"black",
   "stroke_width":5,
   "stroke_dasharray":1,
    "text_color":"black"
},
...
]
```

3.4 Creazione della visualizzazione

JLO è il cuore del progetto, quel file essenziale contenente tutte le informazioni necessarie alla creazione del grafo. Vediamo ora nel dettaglio come è stato implementato lo script che sfrutta i dati nel file JLO.

D3.js

D3, o meglio, Data-Driven Documents [1] è una libreria JavaScript per creare visualizzazioni dinamiche ed interattive partendo da dati organizzati. Per fare ciò si serve degli standard del web come SVG, HTML5 e CSS. La libreria D3, incorporata in una pagina HTML, utilizza funzioni JavaScript prefatte per selezionare elementi del DOM, creare elementi SVG, aggiungere uno stile grafico, oppure transizioni, effetti di movimento e tooltip. Questi oggetti possono essere poi personalizzati utilizzando un foglio di stile CSS. In questo modo grandi collezioni di dati possono essere facilmente convertiti in oggetti SVG usando funzioni di D3 e così generare ricche rappresentazioni grafiche di numeri,grafici, testi, mappe e diagrammi. Inoltre, D3 si occupa di fornire delle API (application program interface) per simulare una forza fisica applicabile agli elementi SVG e di mostrarne gli effetti. I dati di cui D3 ha bisogno possono essere codificati in diversi formati tra cui JSON, CSV e geoJSON [7] [2].

3.4.1 Princìpi di funzionamento

Selezione

Uno dei punti chiave di D3 è la selezione degli elementi del DOM. Questa tecnica consente al programmatore di manipolarli in modo simile a jQuery e effettuare delle operazioni, per esempio, assegnare delle classi o inserire delle proprietà CSS. Una volta selezionato l'elemento, questo può essere rimosso, aggiunto oppure può essere utile per estrapolare delle informazioni. La selezione può basarsi su tag, classi, identificatori, attributi o una posizione nella gerarchia del DOM. Questo processo di modifica, creazione o rimozione degli elementi HTML può essere fatto dipendere dai dati che gli forniamo, per esempio nel JSON, che è il concetto base di D3. [2]

[1] https://github.com/d3/d3/wiki
[2] https://en.wikipedia.org/wiki/D3.js

```
d3.selectAll("p")              // seleziona tutti gli elementi <p>
   .style("color", "lavender") // imposta il colore su "lavender"
   .attr("class", "squares")   // attribuisce la classe "squares"
   .attr("x", 50);             // imposta l'attributo x a 50
```

Associazione di dati

Per un uso più avanzato il caricamento dei dati, da un array o da un file JSON come nel nostro caso, guida la creazione degli elementi. D3 carica il dataset, poi per ognuno di questi elementi crea un oggetto con le proprietà associate (forma,colore, valore) e il relativo comportamento, attraverso delle funzioni che estrapolano i dati dal file JSON. L'esempio sottostante mostra come avviene l'associazione dei dati.

```
// Dati
  var data = [
     { nome:"Gianluigi Buffon", ruolo: "portiere",
     stipendio:4, coloreSquadra: "black"},
     { nome:"Paul Pogba",   ruolo: "centrocampista",
     stipendio:12, coloreSquadra: "red" },
     { nome:"Cristiano Ronaldo", ruolo: "attaccante",
     stipendio:22, coloreSquadra: "white" }
  ];
//Crea un contenitore SVG
  var svg = d3.select("#sezione").append("svg")
        .attr("width", 120)
        .attr("height", 120)
        .style("background-color", "#D0D0D0");
// Crea un elemento SVG dai dati
    svg.selectAll("circle") // crea virtualmente dei cerchi
```

```
      .data(data)          // associa i dati
    .enter()              // per ogni riga dell'array...
      .append("circle")  // lega i dati con i cerchi
        .attr("id", function(d) {
         return d.nome }) // imposta l'id in base al nome
        .attr("r", function(d) {
         return d.stipendio  }) // in base allo stipendio
        .attr("stroke",  function(d) {
         if(d.ruolo)=="portiere"{
            return "white";}
          if(d.ruolo=="centrocampista"){
            return "green";}
          if(d.ruolo=="attaccante"){
            return "red";
          }
          });  // colore del bordo in base al ruolo
        .attr("fill",function(d){
         return d.coloreSquadra });   //il colore del cerchio
                                     //in base alla squadra
```

3.4.2 Loadjlo.js - Lo script

Per prima cosa lo script stabilisce le forze da simulare creandone una tra i nodi collegati (imponendo anche la relativa distanza), una di repulsione globale e una di collisione che impedisce ai nodi di sovrapporsi.

```
var simulation = d3.forceSimulation()
    .force("link", d3.forceLink().id(function(d) {
    return d.id; }).distance(250))
    .force("charge", d3.forceManyBody())
    .force("center", d3.forceCenter(w / 2, h / 2))
```

```
    .force("collide",d3.forceCollide(50).iterations(13) );
```

Successivamente viene lanciata la funziona che caricherà i dati JSON all'interno della quale si svolgeranno tutte le operazioni che ne faranno uso.

```
d3.json(dati, function(error, graph) {
  if (error) throw error;
  ...
  ...
  }
```

In primo luogo vengono processati gli array delle classi e dei template. Le classi vengono aggiunte alla pagina HTML nella sezione <style >mentre i template sono salvati in un array locale in modo da averli a disposizione successivamente. Per creare i nodi indichiamo l'array "nodes" dal file JSON e usiamo una selezione come descritto nella sezione precedente creando un gruppo con il tag <g>che servirà a contenere l'oggetto SVG e il testo al suo interno.

```
var node = svg.append("g")
  .attr("class", "nodes")
  .selectAll("g")
  .data(graph.nodes) //qua viene indicato l'array "nodes"
  .enter().append("g")
  ...
```

Lo script seleziona tutti gli elementi dell'array creando per ognuno la rispettiva figura, assegnando il proprio id, attributi di stile, funzione di "mouseover" ed eventualmente la classe di appartenenza. D3 si occupa di assegnare una posizione in modo automatico ai nodi in base alle impostazioni della forza fisica date in precedenza.
La stessa cosa viene fatta per i collegamenti.

```
var link = svg.append("g")
    .attr("class", "links")
    .selectAll("g")
    .data(graph.links)
    .enter().append("g")
    .append("line")
    ...
```

Successivamente vengono impostati per ogni nodo gli attributi che servono a definirne la grandezza. Questa operazione viene fatta con una selezione diversa per ognuna delle figure poiché richiedono attributi specifici. Per esempio il rettangolo, così come per l'immagine PNG/JPG, ha bisogno di una "x" e una "y" che identificano le coordinate del vertice in alto a sinistra, un attributo "height" che definisce l'altezza e un altro "width" che ne definisce la larghezza, mentre un esagono è una polilinea che ha bisogno di punti ben definiti per essere disegnato. Il cerchio invece ha bisogno solo della lunghezza del raggio oltre che alle coordinate "cx" e "cy" che ne individuano il centro. L'ellisse necessita di un attributo "rx" e un "ry", che definiscono la lunghezza del raggio rispettivamente sull'asse x e y, un "cx" e un "cy" per posizionare il centro della figura. Se la forma del nodo è un template personalizzato lo script ne recupera il nome e le relative informazioni in un array processato all'inizio dello script .
Per appendere del testo nelle figure viene creato un clip-path, che serve per creare un oggetto e mostrarlo in una determinata regione, al cui interno viene scritto del testo. A questo elemento viene assegnato l'id del nodo in modo da legarlo con esso e imporre che rimanga sempre al suo interno. Una volta aggiunto il testo ai nodi e alle etichette sugli archi di collegamento, vengono aggiunti i tooltip con il rispettivi testi all'interno. Successivamente vengono assegnate le coordinate della sorgente e della destinazione ai link e viene aggiunta la funzione "transform" che si occupa ci gestire lo spostamento dei nodi e delle etichette come possiamo vedere nel frammento sottostante.

```
function ticked() {
  link
      .attr("x1", function(d) {
      if(d.direction=="forward")
        return (d.source.x);
      else
       return (d.target.x)
   })
      .attr("y1", function(d) {
      if(d.direction=="forward")
          return (d.source.y)
      else
       return (d.target.y )
   })
      .attr("x2", function(d) {
        if(d.direction=="forward")
          return (d.target.x)
       else
        return (d.source.x) })
      .attr("y2", function(d) {
        if(d.direction=="forward")
          return (d.target.y)
        else
         return (d.source.y)
         });

d3.select(".nodes").selectAll("g")
.attr("transform", function (d) {
   return "translate(" + d.x + "," + d.y + ")";
    });
```

```
    d3.selectAll(".labels").selectAll("g")
.attr("transform", function (d) {
 return
 "translate(" + (d.source.x+d.target.x)/2 + ",
 " + ((d.source.y+d.target.y )/2)+ ")";
   });
 }
```

Il codice integrale dello script si può trovare all'indirizzo
https://github.com/LucaBonini/JLO/blob/master/js/loadjlo.js.

3.5 JLO: esempi di visualizzazioni di dati eterogenei

Dopo aver descritto il formato JLO e lo script che genera il grafo nelle sezioni precedenti, mostreremo degli esempi di visualizzazioni di dati estratti da DBpedia. Premetto che, non avendo sviluppato le prime due fasi del progetto descritto in questo capitolo, i seguenti esempi sono stati realizzati plasmando manualmente il file JLO, anche se ho scritto uno script Pyhton basilare, che esegue la query e crea i file, reperibile al seguente indirizzo:
https://github.com/LucaBonini/JLO/tree/master/PythonToJLO

Esempio 1: Calciatori e squadre

Nella figura 3.5 viene mostrato un grafo orientato contente i giocatori Gianluigi Buffon, Paul Pogba e Cristiano Ronaldo e le relative squadre in giocano o hanno giocato [3].
Sono state impostate le seguenti regole per il layout:

- I nodi sono dei cerchi con i bordi neri non tratteggiati.
- I predicati sono delle etichette blu posizionate su linee gialle che vanno dal soggetto in direzione dell'oggetto.
- I calciatori vengono rappresentati da un immagine fornita dall'utente.
- Le squadra che militano nella prima divisione del relativo campionato nazionale sono di colore arancione.
- La grandezza del cerchio è relativa al numero di titoli vinti.
- Gli archi che collegano una squadra a più giocatori sono rossi e tratteggiati.

Esempio 2: Membri di una band

Nella figura 3.6 viene mostrato un grafo contenente delle band musicali e i componenti che ne hanno e/o ne fanno ancora parte.
Sono state impostate le seguenti regole per il layout:

- Assegniamo ai nodi che rappresentano le band un template comune. Questo è composto da un rettangolo giallo con il bordo tratteggiato nero.
- Rappresentiamo i membri della band come dei cerchi della stessa grandezza e assegniamo una classe comune a tutti che li rappresenti colorati di rosa con un bordo nero.

[3]Demo disponibile all'indirizzo http://eelst.cs.unibo.it/lbonini/

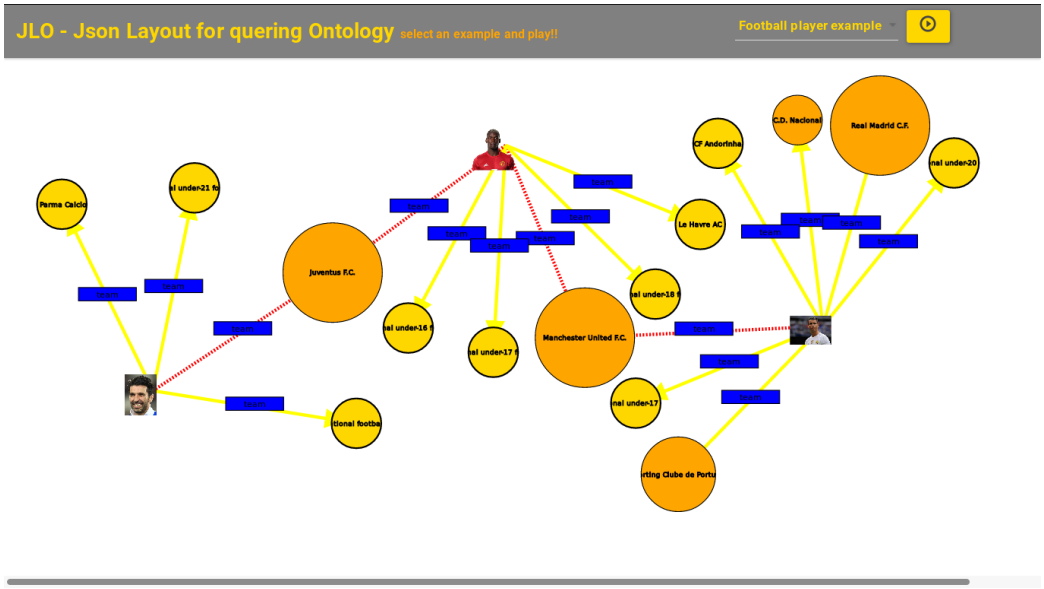

Figura 3.5: JLO esempio 1: si noti che non si vedono le frecce direzionali. Questo è un problema implementativo noto che verrà risolto negli sviluppi futuri.

- I link sono linee nere non tratteggiate per coloro che hanno fatto o fanno parte di una sola band, mentre sono linee rosse tratteggiate per gli altri.
- Le etichette sono dei rettangoli arancioni.

Esempio 3: Autori di libri

Nella figura 3.7 viene raffigurato il grafo contenente gli autori e i libri da loro scritti.
Le regole impostate in questo layout non seguono una logica precisa ma servono per mostrare ulteriori le rimanenti feature che JLO mette a disposizione. In particolare notiamo:

- Alcuni nodi sono stati rappresentati con dei rettangoli, altri con esagoni ed ellisi.

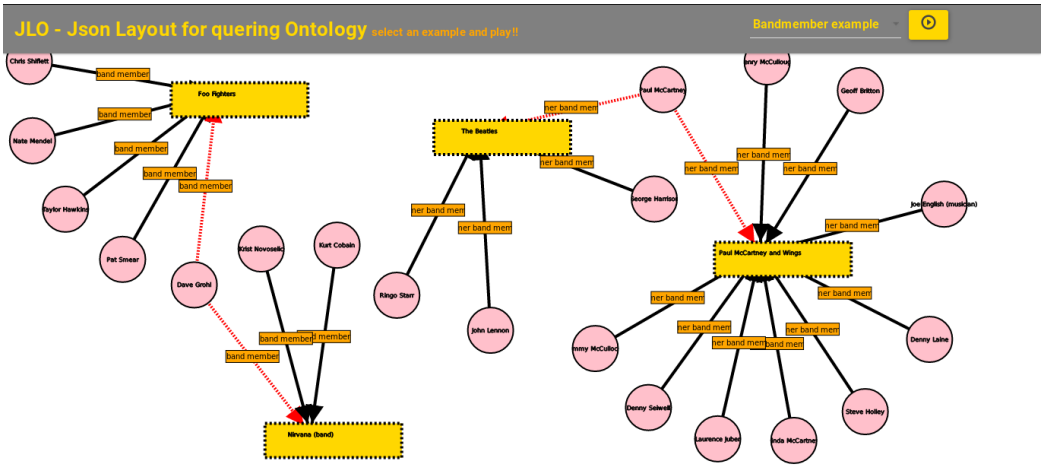

Figura 3.6: JLO esempio 2: anche in questo esempio non si vedono le frecce direzionali. Questo è un problema implementativo noto che verrà risolto negli sviluppi futuri.

- I nodi rappresentati come ellissi hanno il bordo tratteggiato.
- Il testo contenuto nel nodo è presente in colori diversi, così come il colore di riempimento delle etichette.
- Sono presenti nodi con il colore del bordo blu.
- Sono presenti frecce in direzioni diverse.

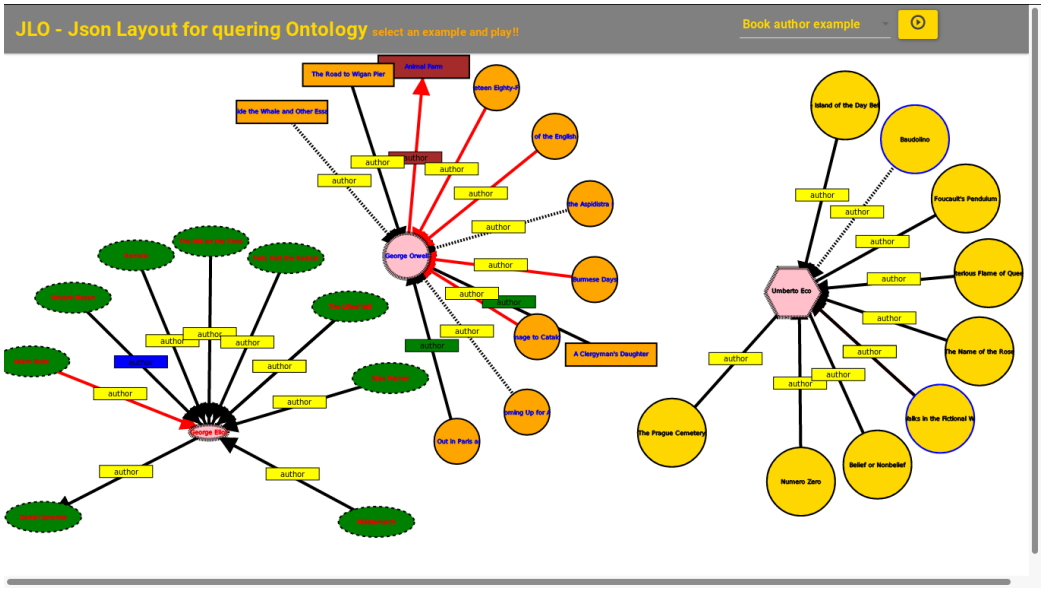

Figura 3.7: JLO esempio 3: la scelta di queste regole vuole solo essere esplicativa di tutte le possibili personalizzazioni concesse dal formato JLO.

Capitolo 4

Valutazione

Vediamo ora in figura 4.1 una tabella che mette a confronto i software visti fino ad ora con la web app basata su JLO descritta nel capitolo precedente. Verranno evidenziati gli aspetti grafici e le feature a disposizione.

	GrOWL	RDF Gravity	WebVOWL	MEMOgraph	Web app JLO based
Piattaforma di sviluppo	Java	java	Html, Css, Javascript	Java	Html, Css, Javascript
Layout grafico	Grafo diretto	Grafo diretto	Grafo diretto	Grafo indiretto	Grafo diretto Grafo indiretto
Rappresentazione Nodi	Ellissi	Rettangoli, Triangoli	Cerchi	Immagini	Ellissi, Esagoni, Rettangoli,Cerchi, Immagini
Personalizzazione Nodi	No	No	No	No	Grandezza,Colore, Colore bordo, Tratteggiamento bordo, Colore testo
Personalizzazione Archi	Archi predefiniti In base alla relazione Tra nodi	No	Archi predefiniti In base alla relazione Tra nodi	No	Spessore, Colore, Tratteggiamento, Direzione
Personalizzazione Etichette sugli archi	Etichette non Presenti	Etichette non Presenti	Etichette presenti ma Non personalizzabili	Etichette non Personalizzabili	Colore, Colore bordo, Colore testo, Spessore bordo, Tratteggiamento bordo
Interattività	Drag and drop nodi	No	Drag and drop dei nodi, Forza fisica simulata, Cambio colore	No	Drag and drop dei nodi, Gravità, attrazione,repulsione, Cambio colore, Cambio spessore bordo, Popup, Click
Query user friendly	No	No	Si	Si	Si*
Ricerca e filtro	Si	Si	Si	Si	Si*
Visualizzazione Tbox	Si	Si	Si	No	No
Visualizzazione Abox	si, in una sezione a Parte non nel grafo	Si	Si	Si	Si
Tipologia applicazione	Desktop	Desktop	Web	Desktop	Web

Figura 4.1: Tabella di confronto

Possiamo notare che la web app basata su JLO possiede tutte le caratteristiche in comune con gli altri strumenti tranne la visualizzazione delle TBox che non è stata implementata poiché il software è stato pensato con il solo scopo di mostrare le asserzioni contenute nel dataset. Questo strumento garantisce una maggior personalizzazione del grafo rispetto a tutti gli altri strumenti in circolazione. La possibilità di disegnare i nodi, gli archi e le etichette secondo un proprio ragionamento è il punto di forza del software. Inoltre l'implementazione basata sulle tecnologie html, css e javascript rende il tool completamente indipendente dal tipo di device e dal sistema operativo impiegato per l'utilizzo, prestandosi quindi anche all'uso da un comune browser su tablet e smartphone. Il codice dello script che prende in input un file JLO è stato testato con tre esempi reperibili all'indirizzo http://eelst.cs.unibo.it/lbonini che ne verificano il corretto funzionamento. Nello specifico:

- Nell'esempio dei calciatori viene testato:
  -L'uso di immagini come nodi
  -La grandezza dei nodi
  -Il colore dei nodi
  -Layout dei link
  -Cambio di colore del nodo al passaggio del mouse

- Nell'esempio delle band musicali viene testato:
  -L'uso dei template per i nodi rappresentanti le band
  -Il cambio di spessore al passaggio del mouse
  -L'assegnamento di classi ai nodi che rappresentano i componenti

- Nell'esempio degli scrittori viene testato:
  -L'utilizzo delle restanti forme come rettangoli, ellissi e esagoni
  -Layout delle etichette
  -Il colore del testo all'interno del nodo
  -Visualizzazione del commento nel popup al passaggio del mouse
  -Il colore e tratteggiamento dei bordi del nodo

Conclusioni e sviluppi futuri

In questo documento abbiamo visto quanto può essere utile rappresentare i dati e quanto sia difficile mostrarli in modo da soddisfare le esigenze di tutti i tipi di utente, da quello più esperto a quello con poca dimestichezza in questo ambiente. I dati semantici essendo predisposti per essere interpretati da una macchina sono di per sè tutt'altro che "user friendly" per l'uomo. Abbiamo visto esserci un gran numero di strumenti in circolazione, molti dei quali però si focalizzano solo sulla rappresentazione strutturale dell'ontologia senza interessarsi delle asserzioni contenute all'interno, altri invece svolgono questa funzione senza dare modo all'utente di intraprendere un processo creativo che lo conduca a una migliore comprensione dei risultati ottenuti. Per questo motivo il progetto di una web app basata su JLO nasce con l'intento mettere nella condizione ottimale l'utente per sfruttare il Semantic Web come strumento di ricerca e visualizzazione come visto nel capitolo 3. JLO è un formato progettato per essere letto da una macchina ma anche dall'uomo grazie alla sua struttura JSON. Ha ampi margini di miglioramento in quanto si potrebbero aggiungere nuovi attributi di stile al layout e altre funzionalità di interazione, oltre che nuove figure per i nodi e nuovi tipi di archi per i link. Sono noti i seguenti problemi legati allo script che genera il grafo:

- I link puntano al centro della figura, perciò se questa supera una certa grandezza nasconde la freccia direzionale.

- Non è ancora stata implementata la freccia bidirezionale ma il formato JLO è gia predisposto per questo.

- I nodi non vengono disposti ancora in modo ottimale.
- Non è ancora possibile disegnare un nodo all'interno di un altro nodo.

Gli sviluppi futuri che andranno a completare il progetto della web app prevedono:

- La creazione di un interfaccia interattiva che consenta di eseguire delle richieste evitando di imparare un linguaggio specifico.
- La personalizzazione del layout del grafico finale senza dover manipolare JLO.
- Implementazione di un pannello per la ricerca e un filtro dei nodi.

Bibliografia

[1] L. Balzer, M. T. Do, and D. Maseluk. Comparison and evaluation of ontology visualizations. 2015.

[2] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011.

[3] F. Ghorbel, N. Ellouze, E. Métais, F. Hamdi, F. Gargouri, and N. Herradi. Memo graph: An ontology visualization tool for everyone. *Procedia Computer Science*, 96:265–274, 2016.

[4] D. A. Keim. Information visualization and visual data mining. *IEEE transactions on Visualization and Computer Graphics*, 8(1):1–8, 2002.

[5] S. Krivov, R. Williams, and F. Villa. Growl: A tool for visualization and editing of owl ontologies. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):54–57, 2007.

[6] S. Lohmann, V. Link, E. Marbach, and S. Negru. Webvowl: Web-based visualization of ontologies. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 154–158. Springer, 2014.

[7] S. Murray. *Interactive data visualization for the Web.* " O'Reilly Media, Inc.", 2013.

[8] O. Noppens and T. Liebig. Interactive visualization of large owl instance sets. In *Proc. of the Third Int. Semantic Web User Interaction Workshop (SWUI 2006), Athens, GA, USA*, 2006.

Ringraziamenti

Vorrei ringraziare la mia famiglia per avermi supportato e per aver sempre creduto in me, tutti gli amici che mi sono stati accanto in questo percorso, il Dott. Poggi e il Prof. Ciancarini.