ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

FACOLTA' DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

TESI DI LAUREA

in
Reti Di Calcolatori LS

Linked Open Data per la pubblica amministrazione: conversione ed utilizzo dei dati

CANDIDATO
Raffaele Ianniello

RELATORE
Chiar.mo Prof.
Antonio Corradi
CORRELATORI
Ing. Luca Foschini
Ing. Mauro Lenzi

ANNO ACCADEMICO 2012/2013
I sessione

Sommario

Indice delle figure

Introduzione

La produzione e il consumo di dati e informazioni sono attività che riguardano una grandissima fetta della popolazione mondiale. Ogni giorno sono prodotti dati da persone o da entità giuridiche, quali società o pubblica amministrazione, anche in maniera inconsapevole. Il solo muoversi con il proprio telefono cellulare in tasca che si connette a una data cella del proprio gestore di telefonia anziché un'altra, oppure il registrarsi all'anagrafe di un comune per il cambio di residenza, o ancora il reddito prodotto in un dato anno fiscale sono solo alcuni degli esempi di azioni abbastanza comuni nella vita di una persona che generano informazioni mantenute dagli enti che forniscono tali servizi.

Queste e altre informazioni possono essere messe in relazione tra di esse per permettere l'estrapolazione e la derivazione di nuove informazioni o per prevedere necessità future.

In questo ambito si inseriscono le tecnologie del web semantico che, grazie alla capacità di fornire un modello per la strutturazione dell'informazione, permettono di unire informazioni prodotte e mantenute presso diverse sorgenti facilitando il compito di analisi dei dati.

Unito a ciò, è in corso presso le pubbliche amministrazioni dei paesi occidentali in particolare, e quindi anche dell'unione europea e italiana, la pratica di "liberare", almeno in parte e in modo aggregato, informazioni in loro possesso della natura più varia e che spaziano dai dati sulla popolazione ai dati del territorio e molti altri. Questa tendenza, sostenuta e stimata dai cittadini stessi e da personalità importanti e di prestigio internazionale, fa sì che i dati della pubblica amministrazione, prodotti a partire proprio dai comportamenti e dalle scelte delle persone, nonché finanziati attraverso le tasse pagate dai contribuenti, ritornino a essere di proprietà di tutti con l'obiettivo sia di trasparenza che di riutilizzo degli stessi.

Uno dei motivi principali di questa apertura è la volontà di creare sviluppo economico a partire da queste informazioni ipotizzando la costruzione di nuovi servizi basati sull'uso dei dati stessi.

Ad esempio, i dati possono essere utilizzati per sviluppare modelli di business per un determinato servizio o prodotto, sulla base della composizione di un ipotetico bacino di utenza. Tale bacino di utenza può essere individuato sulla base dei dati sulla composizione della popolazione resi pubblici dalle amministrazioni comunali, naturali possessori di tali informazioni. O ancora, utilizzare dati geografici pubblicati da pubbliche amministrazioni per fornire informazioni e servizi basati sulla conoscenza di tali dai, come la distanza e il percorso per raggiungere la fermata dei mezzi pubblici più vicina.

L'obiettivo di questo progetto sarà cercare di individuare una metodologia per trasformare i dati messi a disposizione dalle pubbliche amministrazioni e realizzare strumenti che ne rendano l'uso più facile ed intuitivo anche all'utente meno esperto. Il lavoro svolto verrà realizzato in collaborazione con e-Soft, azienda di software di Bologna e il comune di Bologna che sta valutando se e come pubblicare i propri dati anche in formati Linked Data. Ovvero un formato che sottende un modello a grafo interconnesso (contrapposto ad un modello a tabella) che permette di collegare i dati stessi, anche a dati prodotti e pubblicati da enti diversi ed in grado di essere analizzati ed esplorati anche da macchine.

All'interno di questo documento saranno presentate le varie fasi affrontate e saranno mostrati alcuni dei risultati finali ottenuti.

Nel primo capitolo saranno presentati alcuni concetti di base su Linked Data, Open Data e Sistemi Informativi Territoriali; concetti che serviranno per tutto il resto discorso. Nel secondo capitolo saranno presentati alcuni esempi di lavori precedenti realizzati che saranno presi come base per sviluppare il nostro lavoro. Nel terzo capitolo saranno discussi alcuni dettagli tecnici sulle specifiche tecnologie e strumenti utilizzati. Il capito quattro contiene una descrizione del processo svolto che ci ha consentito di realizzare i risultati mostrati nel capitolo cinque. Nell'ultimo capitolo sarà valutato l'approccio scelto.

1 Concetti di base

La disponibilità e il rilascio di Open Data è un patrimonio prezioso per la società civile e per le imprese, ma affinché si possa valorizzare del tutto l'informazione, l'apertura da sola non basta. È desiderabile rendere gli Open Data autodescrittivi e poter inferire conoscenza dall'aggregazione e correlazione di dataset differenti.
Occorre inoltre favorirne la facilità di uso, il reperimento e il consumo sia per gli esseri umani che per i software automatici. In tempi recenti diversi sforzi di ricerca e sviluppo hanno identificato nelle tecnologie del Web Semantico, e in particolare nel modello dei Linked Data, interessanti opportunità per superare le limitazioni dei modelli Open Data. Mentre in generale gli Open Data abbattono le barriere culturali, legali ed economiche al riuso, il movimento Linked Data si concentra piuttosto sulla messa a punto di strumenti che permettono di dare ai dati (aperti o non) un'identità e di renderli collegati tra loro e interoperabili.
In oltre, molti dati sono in qualche modo legati a una specifica area geografica o zona, per cui può risultare utile, anche in ambito di presentazione dei dati, legare i dati ad un area su una mappa.
In questo capitolo verranno trattati i concetti di base del Web semantico, dei dati aperti, o Open Data, e della rappresentazione geografica dei dati.

1.1 Web Semantico

Il Web è un sistema di documenti ipertestuali interconnessi, accessibili attraverso Internet [1]. Questa collezione di ipertesti è stata costruita ad uso prettamente umano. Le informazioni contenute sono sì leggibili da macchine, ma non comprensibili per loro. Infatti queste sono in grado di leggere le informazioni ma non sono in grado di comprenderle, e quindi di elaborarle. Una soluzione proposta, al fine di superare questo limite, è stata quella di utilizzare i *metadati* [2] per descrivere i dati contenuti nel Web. I metadati sono "dati su altri dati", nel caso specifico "dati che descrivono risorse Web". Lo scopo del loro utilizzo è di permettere l'elaborazione automatica delle risorse Web descritte aggiungendo loro una semantica. La semantica, quindi,

aggiunge informazioni che descrivono il significato della risorsa, e ne descrive anche la relazione con altre risorse. In questo modo viene introdotto un collegamento anche tra gli stessi dati presenti nel Web creando, così, una rete dei dati costituente un qualcosa di simile ad un database globale. Ciò permette di combinare dati provenienti da diverse applicazioni presenti sul Web. Il Web semantico, quindi, non è un Web separato da quello che tutti conosciamo, ma una estensione dello stesso, dove ad ogni informazione è associato un significato preciso, in modo da permettere una cooperazione migliore tra computer e persone.

Questa estensione, denominata *Semantic Web* e sviluppata dal *World Wide Web Consortium* (W3C) [3], fornisce una standardizzazione del modo in cui devono essere rappresentate le relazioni e le regole di ragionamento sui dati, e permettere di esportare sul Web regole di altri sistemi di rappresentazione della conoscenza. Lo scopo di questa standardizzazione è di facilitare la condivisone dell'informazione tra applicazioni diverse.

1.1.1 Linked Data

Un fattore chiave per poter far si che i dati siano facilmente riusabili è che siano ben strutturati. Migliore è la definizione della struttura, maggiore è la facilità con cui è possibile creare strumenti per poterli riutilizzare.

Il Web Semantico ha come obiettivo di creare collegamenti tra i dati, non solo tra le pagine web, in modo che, a partire da alcuni dati, possono esserne esplorati altri collegati ai primi. I *Linked Data* descrivono un metodo per pubblicare dati strutturati in modo tale da poter essere interconnessi con altri dati attraverso i metadati [4]. La struttura dei dati che si viene a creare grazie alla presenza di tali interconnessioni, identificata comunemente come "*Web of data*" (**Figura 1.1**), può così essere facilmente navigata anche da macchine.

I Linked Data sono stati proposti da Tim Berners-Lee come modo per creare una rete di dati interoperabile, prendendo esempio dalla interconnessione dei documenti nel Web ipertestuale. Berners-Lee consiglia di utilizzare 4 principi di base per perseguire questo obiettivo [5]:

- utilizzare URI (*Uniform Resource Identifier*) come nomi per le "cose";

- usare URI per protocollo HTTP in modo che sia possibile cercare e risolvere quei nomi;
- quando qualcuno cerca un URI, fornire un'informazione utile;
- includere link ad altri URI, così da permettere, a chi cerca, di scoprire nuovi collegamenti.

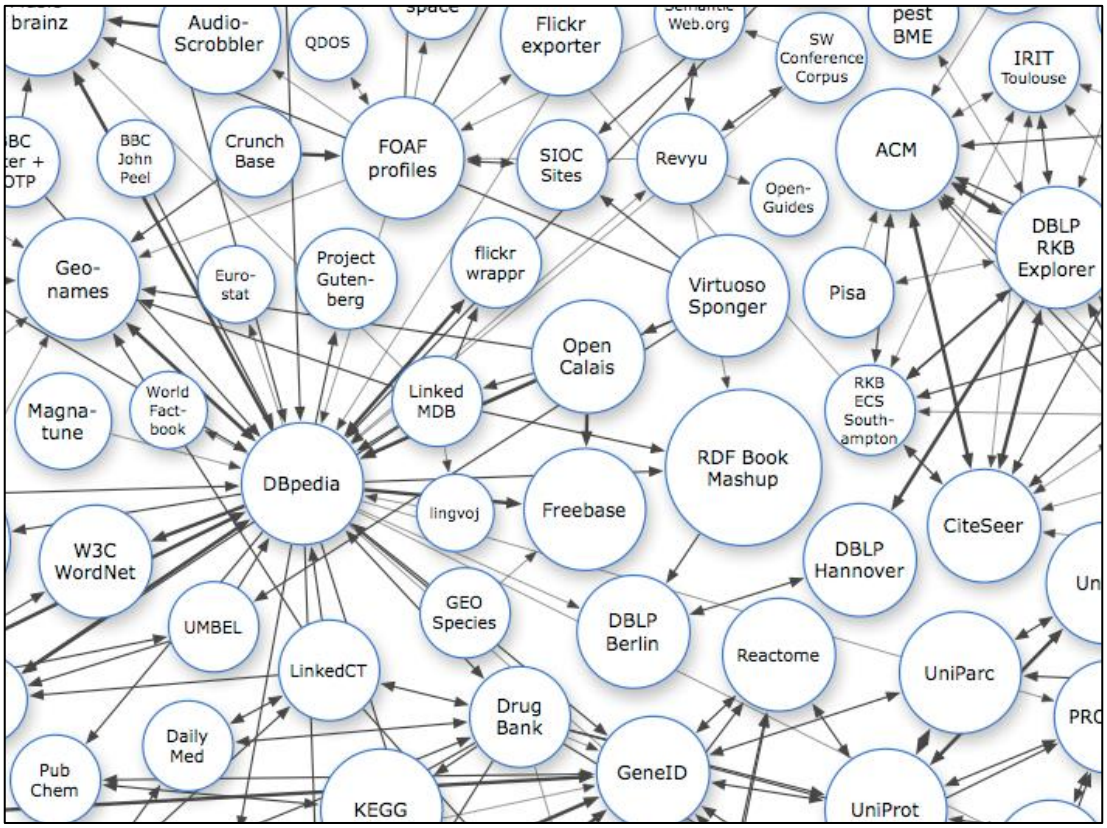

Figura 1.1: parte del diagramma del progetto Linking Open Data (LOD) [6]

I Linked Data, al contrario dei database relazionali che utilizzano le tabelle per rappresentare la conoscenza, organizzano la conoscenza in forma di grafo strettamente interconnesso permettendo una rappresentazione più completa e, allo stesso tempo, più flessibile e più facilmente riorganizzabile in caso di cambiamento dello schema di descrizione della base di conoscenza.

1.1.2 Ontologia

L'ontologia, una delle branche fondamentali della filosofia, è lo studio dell'essere in quanto tale, nonché delle sue categorie fondamentali. In un'accezione più estesa, si intende un'indagine sull'essere al di là degli enti attraverso i quali esso ci si

manifesta nelle apparenze e nei fenomeni: la ricerca dell'Essere o del loro fondamento ultimo.

Il termine ontologia è stato cooptato da ricercatori di intelligenza artificiale e della rappresentazione della conoscenza, per descrivere il modo in cui diversi schemi vengono combinati in una struttura dati contenente tutte le entità rilevanti e le loro relazioni in un dominio.

Un'ontologia per il Web è costituita da una tassonomia e da un insieme di regole di inferenza. La tassonomia definisce classi di oggetti e relazioni tra gli stessi. Per esempio, uno *studente* può essere definito come un tipo di *persona* e una *matricola* può essere definita come applicabile solo a oggetti di tipo *studente.* Classi, sottoclassi e relazioni fra entità sono strumenti molto potenti. Si possono esprimere un grande numero di relazioni fra entità assegnando proprietà a classi e permettendo alle sottoclassi di ereditarle.

In questo modo è possibile combinare informazioni presenti su database diversi memorizzati con identificatori diversi ma che in realtà identificano la stessa categoria logica. Idealmente, una macchina, grazie all'utilizzo di un'ontologia comune, sarebbe in grado di scoprire e individuare questa uguaglianza di significati per qualsiasi database incontrasse.

Tuttavia possono sorgere problemi se si cerca di aggregare informazioni provenienti da due sorgenti diverse che fanno uso di ontologie diverse. Questo problema può essere risolto attraverso l'uso di relazioni di equivalenza in una o entrambe le ontologie.

In ambito di Web Semantico, i concetti sono identificati univocamente con gli URI, mentre i vocabolari [7] dei termini sono utilizzati per descrivere le relazioni tra le risorse. Queste descrizioni rappresentano lo schema dell'informazione e forniscono le indicazioni utilizzate dai Linked Data per rappresentare l'informazione. I vocabolari, a loro volta, sono descritti sotto forma di Linked Data.

Un'ontologia usa un certo numero di elementi di base per descrivere le relazioni tra i dati. L'espressività di un'ontologia varia in funzione del numero di tali elementi, per cui è possibile ottenere una descrizione della struttura dei dati con complessità diverse.

Le regole di inferenza forniscono un ulteriore potenza. Ovvero di estrarre nuova informazione a partire dai dati disponibili. Ma per eseguire tale processo risulta necessario un *query engine*.

1.1.3 Query engine

Un ontologia può esprimere una regola per cui "se una matricola è associata ad uno studente e uno studente è una persona a cui è associato un indirizzo di residenza, allora ad una matricola è associato un indirizzo di residenza". Un programma può dedurre che tutta la comunicazione burocratica con lo studente con determinata matricola possa essere inviata per posta all'indirizzo di residenza della persona associata. Una macchina non comprende alcuna di queste informazioni ma è in grado di manipolarne i termini in modo da avere un significato per gli utenti umani.

Un *query engine* (motore per esecuzione di interrogazioni) è un sistema in grado di processare una specifica richiesta detta *query* effettuata in un linguaggio standard e utilizzata sia nei database relazionali sia nelle basi di conoscenza di tipo linked data. È in grado anche di ricavare informazioni a partire da dati in formato linked e descritti da un ontologia. Per ottenere quest'ultimo risultato è necessario che il query engine utilizzi un *Rule-based Reasoner* (ragionatore basato su regole), ovvero un sistema in grado di estrarre nuova informazione a partire da informazioni strutturate applicando esaustivamente un insieme di regole. Tali regole sono dette regole di inferenza e sono applicate alle informazioni iniziali della base di conoscenza in formato linked data per elaborare ed estrarre nuovo significato dai dati iniziali.

Queste regole possono essere applicate un numero arbitrario di volte anche con le nuove conclusioni prodotte alla fine del passo di inferenza.

Per esempio possiamo ipotizzare di avere a che fare con una ontologia composta da un concetto *Veicolo* con una struttura gerarchica di altri concetti come *Automobile* e *Motociclo,* e di una base di dati facente uso di tale ontologia, però composta solo da istanze di automobili e motocicli. Ad una richiesta dell'utente che prevedesse di elencare tutti i *veicoli* memorizzati in un query engine senza un reasoner risponderebbe alla query con una risposta vuota. Al contrario l'utilizzo di un reasoner permetterebbe di trovare il risultato corretto, ovvero la lista completa di tutte le istanze di *Automobile* e *Motociclo*.

1.2 Open Data

I dati normalmente sono parti di una informazione o conoscenza strutturata che può essere codificata e archiviata in un formato digitale. Esempi pratici sono i dati in possesso delle pubbliche amministrazioni, come atti ufficiali o le informazioni geografiche, le statistiche e i dati ambientali.

Con il termine *Open Data* (dati aperti, di libero utilizzo) si indica l'operazione di rendere accessibile tali dati a chiunque, abbattendo, per quanto possibile e ragionevole, le restrizioni tecnologiche ed imponendo vincoli legali minimi al riuso dei dati.

Gli open data, e in particolare gli *open government data*, sono una immensa risorsa ancora in gran parte inutilizzata. Molte persone e molte organizzazioni raccolgono, per svolgere i loro compiti, una vasta gamma di dati diversi. Quello che fa il governo è particolarmente importante in questo senso, non solo per la quantità e centralità dei dati raccolti, ma anche perché la maggior parte dei dati governativi sono pubblici per legge, e quindi dovrebbero essere resi aperti e disponibili all'uso per chiunque. Ci sono molte circostanze in cui possiamo attenderci che i dati aperti abbiano un valore rilevante e ci sono anche numerose categorie di soggetti e organizzazioni che possono trarre beneficio dalla disponibilità di dati aperti, inclusa la pubblica amministrazione. Ad esempio, in Canada, i dati aperti hanno fatto risparmiare 3,2 miliardi di dollari in un caso di frode fiscale legato alla beneficenza. Molti siti, tra cui l'italiano openparlamento.it, tracciano le attività dei parlamenti e il processo di formazione delle leggi, in modo da mostrare cosa succede esattamente e quali parlamentari sono coinvolti nelle varie attività.

Nuove combinazioni di dati possono creare nuova conoscenza e nuove intuizioni, che possono portare a campi di applicazione inimmaginabili. Tali eventi sono già accaduti in passato, ad esempio quando il dottor John Snow scoprì la correlazione tra l'inquinamento dell'acqua potabile e il colera nella Londra dell'800, combinando i dati sui morti per colera del quartiere Soho con quelli sull'ubicazione dei pozzi dello stesso quartiere. Tale scoperta portò alla decisione di costruire un sistema fognario a Londra, migliorando di molto le condizioni generali di salute della popolazione. Probabilmente vedremo di nuovo nascere intuizioni simili dalla combinazione di insiemi diversi di dati aperti.

La disponibilità e il rilascio di Open Data è, quindi, un patrimonio prezioso per la società civile e per le imprese, ma affinché si possa valorizzare del tutto l'informazione, l'apertura da sola non basta. È desiderabile rendere gli Open Data autodescrittivi e poter inferire conoscenza dall'aggregazione e correlazione di dataset differenti. Occorre inoltre favorirne la facilità di uso, il reperimento e il consumo sia per gli esseri umani che per i software automatici.
Le tecnologie del Web Semantico, e in particolare nel modello dei *Linked Open Data* risultano in grado di fornire opportunità per superare le limitazioni dei modelli Open Data. Mentre in generale gli Open Data abbattono le barriere culturali, legali ed economiche al riuso, il movimento Linked Data, come detto in precedenza, si concentra piuttosto sulla messa a punto di strumenti che permettono di dare ai dati (aperti o non) un'identità e di renderli collegati tra loro e interoperabili.

1.2.1 Linked Open Data

Un Linked Data non deve essere necessariamente *open*, molti dati sono usati come linked in contesti privati per uso personale o aziendale. Se però viene dato clamore ad una operazione di "liberazione" dei dati sotto formato linked, questa operazione deve rispettare delle regole. In base a tali regole è possibile anche classificare la qualità dei dati prodotti in funzione della facilità di utilizzo e dell'aderenza ai principi del web semantico. Tale classificazione, proposta per la prima volta da Tim Berners-Lee nel 2006 [5] prevede di assegnare:

- 1 stella: quando il dato è messo a disposizione di tutti sul web con licenza libera.
- 2 stelle: quando il dato è presente sotto forma di struttura dati leggibile automaticamente (es. una tabella di un foglio di calcolo).
- 3 stelle: quando il dato è presente come nel caso precedente ma in formato non proprietario.
- 4 stelle: quando il dato del caso precedente usa gli standard Linked Data del W3C per identificare i dati
- 5 stelle: quando i dati sono collegati ad altri, utilizzando gli standard W3C, per fornire un contesto.

Come si vede da questa classificazione è importante che i dati abbiano sia una licenza che un formato di tipo aperto.
L'utilizzo di un formato non proprietario per la memorizzazione dei dati è importante perché non vincola l'utilizzatore all'uso di strumenti di lettura e visualizzazione specifici e spesso anche costosi.
La licenza da associare ai dati aperti gioca un ruolo importante sul futuro dei dati stessi: da ciò può dipendere la capacità e la spinta verso l'utilizzo di tale dataset operata da diversi attori e intenzionati ad attuare diversi modelli di business a partire da tali dati. Infatti, la scelta di una licenza che limiti il riuso dei dati nella realizzazione di prodotti derivati, e quindi nell'estrapolazione di informazioni con il confronto con altri dataset, potrebbe non avere molto senso nell'ottica dell'apertura stessa dei dati. Importante è anche la scelta di una licenza riconoscibile in vasti ambiti. Da questo punto di vista un importante standard *de facto* è il mondo delle licenze Creative Commons [8], proposte per la prima volta nel 2001 dal giurista statunitense Lawrence Lessig, e che hanno conosciuto una diffusione a livello internazionale.
I due gradini più alti di questa classificazione prevedono proprio l'utilizzo di strumenti del web semantico per la pubblicazione dei dati. I dati pubblicati sotto questi formati possono favorire l'interoperabilità facilitando il lavoro di integrazione tra diverse sorgenti di dati proprio come decritto in precedenza nella sezione 1.1.

1.3 Sistema informativo territoriale (GIS)

Con l'acronimo GIS (*Geographicl Information System*) si è soliti riferirsi ad una categoria di applicazioni che gestiscono informazioni spaziali. Si tratta di un sistema informativo computerizzato che permette l'acquisizione, la registrazione, l'analisi, la visualizzazione e la restituzione di informazioni derivanti da dati geografici (geo-riferiti).
L'aspetto che caratterizza il GIS è quello geometrico: esso memorizza la posizione del dato impiegando un sistema di proiezione reale che definisce la posizione geografica dell'oggetto. Il GIS gestisce contemporaneamente i dati provenienti da diversi sistemi di proiezione e riferimento (es. UTM o Gauss Boaga).

L’obiettivo principale di un sistema GIS è di fornire strumenti per poter unire ed elaborare informazioni provenienti da diverse sorgenti di dati territoriali ad esempio, derivando informazioni topologiche tra figure geometriche, oppure unendo informazioni associate alle geometrie tramite funzioni di *join* spaziale (ad es. trasferire le uniformazioni di un poligono a tutte le figure geometriche contenute). Oppure di effettuare ricerche ed interrogazioni basate sui campi alfanumerici o su aree di selezione.

1.3.1 Sistemi di proiezione e riferimento

Le proiezioni vengono usate in cartografia per rappresentare su un piano (sulle carte geografiche o sugli schermi dei computer) un fenomeno che nella realtà esiste sulla superficie della sfera (o più propriamente di un geoide). È impossibile evitare deformazioni (lo stesso mappamondo ne subisce alcune), ma alcune proiezioni vengono privilegiate per i pregi che presentano.

Le proiezioni cartografiche possono essere costruite (e classificate) in modo da possedere alcune proprietà. Ad esempio una proiezione può essere:

- *equivalente* se mantiene i rapporti tra le superfici, cioè se le superfici sono in scala;
- *equidistante* se mantiene i rapporti tra le distanze da un punto (o da due punti, ma è impossibile costruire carte con tutte le distanze in scala);
- *conforme* (o *equiangola*, o *isogonale*) se mantiene gli angoli.

Importante è anche il modo usato per rendere “piatte” tali rappresentazioni della superficie terrestre. La *proiezione di sviluppo*, una delle principali tecniche di attuazione della proiezione si ottiene per proiezione prospettica su un altro solido (tipicamente un cilindro o un cono), che viene poi sviluppato ("srotolato").

La proiezione Universale Trasversa di Mercatore (abbreviata in UTM, Universal Transverse of Mercator) o "Proiezione Conforme di Gauss" è una delle soluzioni meglio riuscite al problema di rappresentare la superficie terrestre a due raggi di curvatura. Il sistema è basato su una griglia, un sistema cartesiano che si affianca al sistema angolare di latitudine e longitudine. La proiezione UTM si utilizza dal parallelo di 80° sud a quello di 80° nord.

La Terra viene divisa in 60 fusi di 6° di longitudine ciascuno, a partire dall'antimeridiano di Greenwich in direzione Est, l'Italia è quindi compresa tra i fusi 32, 33 e 34.
È sempre possibile trasformare, tramite equazioni matematiche, in maniera precisa una mappa da una proiezione ad un'altra. Esistono tecniche meno precise ma più veloci di trasformazione che prevedono l'uso di griglie di riferimento.

1.3.2 Rappresentazione dei dati

Per la rappresentazione dei dati in un sistema informatico occorre formalizzare un modello rappresentativo flessibile che si adatti ai fenomeni reali. Nel GIS abbiamo tre tipologie di informazioni:

- *geometriche*: relative alla rappresentazione cartografica degli oggetti rappresentati; quali la forma (punto, linea, poligono), la dimensione e la posizione geografica;
- *topologiche*: riferite alle relazioni reciproche tra gli oggetti (connessione, adiacenza, inclusione, ecc.;
- *informative*: riguardanti i dati (numerici, testuali ecc…) associati ad ogni oggetto.

A differenza della cartografia su carta, la scala in un GIS è un parametro di qualità del dato e non di visualizzazione. Il valore della scala esprime le cifre significative che devono essere considerate valide delle coordinate di georeferimento.
Il formato dei dati che un sistema GIS è in grado di gestire può essere classificato in dei tipi principali: immagini raster e immagini vettoriali.
Il tipo di dato raster consiste in una matrice di celle a cui è associato un valore diverso ad ognuna, quindi delle immagini caratterizzate da risoluzione e livelli di colore rappresentabili per ogni cella (o pixel). In generale sono perlopiù ortofoto aeree memorizzate come immagini bitmap ma è possibile anche rappresentare mappe tematiche di vario tipo.
Nella grafica vettoriale, invece, un'immagine è descritta mediante un insieme di primitive geometriche che definiscono punti, linee, curve e poligoni.
La grafica vettoriale, essendo definita attraverso equazioni matematiche, è indipendente dalla risoluzione, mentre la grafica raster, se viene ingrandita o

visualizzata su un dispositivo dotato di una risoluzione maggiore di quella dell'immagine originale perde di definizione. Una linea che percorre lo schermo trasversalmente se viene rappresentata utilizzando la grafica raster viene memorizzata come una sequenza di pixel colorati disposti a formare la linea. Se si provasse ad ingrandire una sezione della linea si vedrebbero i singoli pixel che compongono la linea. Se la medesima linea fosse memorizzata in modo vettoriale la linea sarebbe memorizzata come un'equazione che parte da un punto identificato con delle coordinate iniziali e termina in un altro punto definito con delle coordinate finali. Ingrandire una sezione della linea non produrrebbe artefatti visivi o la visualizzazione dei singoli pixel componenti l'immagine, dato che la linea sarebbe visualizzata sempre con la massima risoluzione consentita dal monitor.

1.3.3 Dati geografici vettoriali

Le informazioni geografiche di tipo vettoriale sono memorizzate in appositi file creati appositamente per contenere questo tipo di informazione. Uno dei formati più popolari è certamente l'ESRI Shapefile.

Il formato è stato sviluppato e regolato da ESRI, allo scopo di accrescere l'interoperabilità fra i sistemi ESRI e altri GIS. Di fatto è diventato uno standard per il dato vettoriale spaziale, e viene usato da una grande varietà di sistemi GIS. Con "shapefile" si indica di norma un insieme di file con estensione `.shp`, `.dbf`, `.shx`, più altri opzionali. Tra gli opzionali è presente un file di proiezione che conserva l'informazione sul sistema di coordinate da usare per visualizzare le geometrie. Spesso con *shapefile* si indica però solo i file "`.shp`". Tuttavia questo file da solo è incompleto poiché interpretazione ed utilizzo dipendono dagli altri file.

Gli shapefile descrivono spazialmente punti, polilinee e poligoni, noti come dati geometrici primitivi, utilizzabili, ad es., per rappresentare lampioni, strade e edifici rispettivamente. A ciascun elemento possono essere associati ulteriori attributi che descrivono le voci (ad es. *nome, numero di piani, numero civico*). Infatti uno dei file di base contiene anche tutti gli attributi associati a ciascuna geometria.

Gli Shapefile non possono contenere informazioni topologiche (adiacenza, connessione, prossimità, coincidenza) sui loro singoli elementi geometrici (noti

anche come *feature*); essi sono quindi la mera rappresentazione spaziale delle feature e degli attributi.
Dal momento che per elaborare uno shapefile non si devono processare anche informazioni di tipo topologico, questi hanno il vantaggio di poter essere visualizzati ed editati più velocemente rispetto ad altre sorgenti di dati.
Un formato alternativo proposto più recentemente dal Open Geospatial Consortium (OGC) è il Geography Markup Language (GML) che consiste in un formato basato su XML il cui scopo non è solo quello di rappresentare informazioni spaziali in modo vettoriale ma anche di essere un formato di interscambio per transazioni geografiche attraverso internet.

1.4 Motivazioni

Il lavoro svolto si inserisce all'interno del processo di apertura dei dati svolto dalle pubbliche amministrazioni di diverso governi, soprattutto occidentali. Ed è consistito nel cercare di analizzare la complessità del processo di transizione dal sistema di pubblicazione attualmente usato: dati in forma tabellare pubblicati su semplici file scaricabili dai siti web delle rispettive amministrazioni, alla creazione di un archivio dei dati in formati Linked Data interrogabili da remoto, senza vincoli sulla modalità di interrogazione, e creando degli strumenti che permettessero di facilitare tale lavoro per gli utenti meno esperti.
Il lavoro di questa tesi si è svolto all'interno di un progetto di cooperazione tra la facoltà di Ingegneria dell'Università di Bologna, il Comune di Bologna e l'azienda di software e-Soft. Ed è partito dall'analisi dei dati messi a disposizione del pubblico dal comune di Bologna sul sito OpenData [9].

2 Dagli Open Government Data ai Linked Data

2.1 Lavori precedenti

Gli Open Government Data (dati aperti della pubblica amministrazione o OGD) stanno diventando di importanza crescente in tutto il mondo, anche se la maggior parte delle iniziative prevedono soltanto di distribuire i dati in formati proprietari su file scaricabili dagli utenti. Stime e sondaggi hanno mostrato che allo stato attuale esistono relativamente poche iniziative che cercano di combinare gli OGD con i Linked Data sul Web [10]. Tra questi spiccano i casi di Data.gov.uk e Data.gov che risultano essere caratterizzati da una inusuale spinta verso l'uso dei Linked Data sul Web.

2.1.1 La lezione di Data.gov.uk

Nel Regno Unito Data.gov.uk [11] è il progetto che rende disponibile ai cittadini dati non personali, in possesso del governo inglese, come open data. Si tratta di un catalogo con migliaia di dataset scaricabili sotto una licenza permissiva. I dati sono spesso presentati sotto forma di CSV (comma separated value [12]), ma la trasformazione dei dati in formato RDF, in corso d'opera, permette di aumentare l'utilità dei dati.

Il progetto, nella prima fase, prevedeva la trasformazione di catalogo limitato di dati in formato RDF, mentre una seconda fase è consistita nel affrontare il problema dell'eterogeneità semantica attraverso l'uso di schemi e ontologie.

La prima traslazione dei dati è stata intenzionalmente limitata e ha preservato l'originale disposizione dei dati. Sono stati presi dati anche da Data.gov e dal catalogo dei dati della pubblica amministrazione australiana, e questo ha rivelato subito un problema di normalizzazione dei dati. Ad esempio il sistema di numerazione per indicare una data era differente tra i diversi dataset ed ha comportato la necessità di dirimere tali conflitti prima di effettuare comparazioni.

Le operazioni di conversione in RDF sono state fatte attraverso strumenti automatici rendendo i dati collegabili attraverso URI ma senza determinarne la semantica. Per

allineare la semantica relativa a termini specialistici è stata impiegata una fase specifica. Fase complessa che è stata costituita principalmente nella corretta identificazione di termini con stesso significato.
L'utilizzo di dati geografici ha fornito un modo intuitivo di allineare i dataset, dal momento che molti dati delle amministrazioni fanno riferimento a specifici territori. L'applicazione sviluppata all'interno di Data.gov.uk ha permesso di dare un contesto ai dati mostrati oltre che la capacità di creare nuove basi di aggregazione dei dati a partire dalla possibilità di collegare luoghi ai dati.
All'interno del progetto è stato creato un sistema per integrare dati ed esplorare gli stessi che non obbligasse gli utenti ad avere conoscenze specifiche di linguaggi di interrogazione.
La visualizzazione dei risultati di interrogazioni è stato un passo importante all'interno del progetto, permettendo di specificare la granularità delle visualizzazioni sulla base di aree regionali e mostrando grafici confrontandoli con regioni limitrofe.

2.1.1.1 Conclusioni

Il lavoro di Data.gov.uk [11] ha messo in luce possibili soluzioni a problemi ma anche domande aperte relativamente all'operato dei governi, delle comunità tecniche e dei cittadini.
Riguardo all'operato del governo, uno dei primi problemi è il tipo di licenza con cui vengono rilasciati i dati, prima ancora dei formato con cui sono pubblicati gli stessi. Le persone preposte alla decisione delle policy con cui devono essere rilasciati i dati devono essere a conoscenza che le licenze sono il maggior ostacolo alla diffusione degli Open Government Data. Inoltre, sono ancora pochi gli enti che mettono a disposizione i dati grezzi e difficilmente relativi a dati di livello regionale.
Dal punto di vista della comunità tecnica vengono messe in evidenza la carenza di strumenti che permetterebbero di facilitare la creazione di dati in formato linked a partire dai dati delle pubbliche amministrazioni anche con possibilità di automatizzare certe operazioni. Un altro problema riguarda l'individuazione di punti di unione tra i dati in modo da permettere ai consumatori di assemblare un'immagine coerente fra i dataset, come ad esempio il carattere geografico dei dati per poterli unire e confrontare.

Dal punto di vista dei cittadini la trasparenza può creare opportunità per mantenere i governi responsabili e creare servizi innovativi. Inoltre i cittadini potrebbero avere la facoltà di correggere errori o mancanze riguardanti cose con cui essi interagiscono o di cui sono esperti (come scuole, strade, fermate di trasposto pubblico, ecc.). Inoltre è necessaria una infrastruttura per permettere alle persone di creare le proprie applicazioni basate su linked data e garantire il massimo beneficio derivante dagli Open Government Data.

I limiti evidenziati relativi al processo di trasformazione e pubblicazione dei dati riguardano:

- la necessità di più portali internazionali organizzati per settore invece di portali con copertura nazionale e malfunzionanti per il recupero dei dati;
- migliori standard per allineare termini ontologici, che emergerebbero più facilmente con la comparsa di più dataset e maggiore trasparenza;
- interfacce di interrogazione più flessibili e personalizzabili;
- importante l'aumento del numero di applicazioni preparate per presentare informazioni ai cittadini. Lo sviluppo di un ecosistema intorno a questa tecnologia sarà vitale per la stessa.

In sostanza il rilascio di Open Government Data non è solo un problema tecnico ma richiede un dialogo tra i possessori dei dati, gli utenti e gli sviluppatori.

2.1.2 Il caso di Semantic.data.gov

Data.gov, il portale open data del governo degli Stati Uniti d'America, è stato concepito come una piattaforma di accesso e pubblicazione di dati. Ma già nel primo anno era chiaro che le persone non erano interessate a generici dati governativi ma ad uno specifico tipo di dati. Perciò sono state istituite diverse comunità con l'intenzione di focalizzarsi su uno specifico settore della pubblica amministrazione, come ad esempio la salute, la pubblica sicurezza o la scuola. Il team di Data.gov fornisce aiuto alle comunità su come distribuire valore ai cittadini, come partecipare alla comunità e quali funzionalità e contenuti includere. La partecipazione è aperta al pubblico e i membri provengono principalmente dal mondo accademico e industriale sia all'interno che all'esterno degli Stati Uniti. I risultati dei lavori delle comunità

include applicazioni, linee guida per permettere migliori decisioni o presentazione di unione di informazioni.

All'interno di Data.gov, la comunità Semantic.data.gov [13] si focalizza esplicitamente nel fornire supporto alle altre comunità per la realizzazione di dataset in formato linked. Tra queste, la comunità che ne fa più uso è quella legata alla salute (Health.data.gov). In particolare è presente un sistema di confronto tra servizi ospedalieri e criteri di valutazione degli ospedali sul territorio nazionale. Gli utenti possono prelevare l'intero dataset oppure esaminare i dati in maniera più fine. Durante la fase di esplorazione è presente una interfaccia che aggrega automaticamente i dati relativi alla selezione. Ad esempio a partire dalla selezione di un ospedale il sistema aggrega tutti i dati relativi all'ospedale selezionato da tutti i dataset disponibili. Inoltre i dati sono disponibili in diversi formati in modo che sia facile costruire, da parte dell'utente, una propria applicazione usando Health.data.gov come piattaforma.

Il portale Energy.data.gov è un secondo esempio di uso di Linked Open Data. In questo caso un requisito deciso dalla comunità è stato di rendere possibile integrare la ricerca tra più sorgenti di dati sull'ambito energetico, oltre a Data.gov. In questo caso l'utilizzo di un modello comune dei metadati per creare un catalogo dei dataset da varie parti del mondo ha permesso di ottenere una possibilità di estendere le capacità con future integrazioni molto più semplice.

Il governo USA ha sempre avuto come obiettivo una collaborazione pubblico-privato per migliorare le tecnologie e i servizi di e-government. E un obiettivo particolare sono sempre state le università e altri gruppi in ambito educativo.

Un caso di particolare successo tra le collaborazioni è stata la partnership con Tetherless World Constellation del Rensselaer Polytechnic Institute dello stato di New York, una "costellazione" di ricercatori di diverse discipline che studiano i principi ingegneristici e scientifici alla base del Web, con lo scopo di estenderne le capacità attraverso lo sviluppo di nuove tecnologie [14].

Tale team ha realizzato specifici contributi nella fase di produzione dei dati, sviluppando un approccio alla conversione dei dataset in RDF e collegandoli con risorse esterne come DBPedia.org, e nel consumo di tali dati, ovvero nella creazione di strumenti per facilitare l'unione di dati diversi e nella visualizzazione dei risultati.

Per lo sviluppo futuro è prevista la creazione di iniziative internazionali per la realizzazione di sistemi software per permettere di abbattere le barriere in ingresso per la creazione di una piattaforma per gli open data. Altre opportunità sono previste da una migliore integrazione di dati geospaziali e funzionalità integrate di in locale di esplorazione e visualizzazione dei dati, oltre che la creazione di un vocabolario specifico per le amministrazioni.

Gli autori dello studio [13] concludono affermando che la scelta di Data.gov di abbracciare le tecnologie del web semantico hanno reso Data.gov stesso rilevante e utile all'interno del processo decisionale nazionale. E, sebbene l'operazione di creare comunità distinte per area di interesse invece di un'unica collezione di dataset sia stata critica ha permesso la creazione di gruppi di discussione intorno ad importanti temi nazionali.

2.1.2.1 Il portale di Tetherless World Constellation per un ecosistema in aiuto alla creazione di Linked Open Government Data

L'approccio del team di Tetherless World Constellation (TWC), che, come detto in precedenza, ha realizzato un portale per la visualizzazione e l'analisi dei dati presenti sul portale Data.gov, al problema degli Open Government Data è scaturito dalla volontà di andare oltre il modello, presente in approcci recenti, di fornire servizi per il recupero dei dati accessibili solo attraverso API (*Application Programming Interface*) messe a disposizione tramite Web Service in cui l'accesso è limitato tramite le interfacce fornite e che possono introdurre costi di gestione non banali [15]. L'approccio scelto è stato proprio di fornire accesso diretto ai dataset in formato Linked Data su file RDF e attraverso endpoint SPARQL, con lo scopo di favorire l'integrazione tra i dati di sorgenti diverse.

Lo scopo del portale di TWC è stato quello di creare un ecosistema in grado di produrre, gestire e consumare dati in modo da servire diverse tipologie di pubblico, a partire dai dipendenti pubblici che si occupano della pubblicazione e gestione dei dati, agli sviluppatori per la creazione di applicazioni basate su tali dati, fino ai semplici cittadini che hanno la necessità di informarsi e vogliono visualizzare i dati.

La fase di produzione dei dati è stata divisa in tre parti:

- stadio di catalogazione: è stato creato un catalogo dei dataset disponibili e identificati i dataset con identificatori unici con metadati associati;

- stadio di recupero: sono state create delle fotografie dei dati in particolari istanti temporali. In modo da indicare correttamente la versione di tali dati e a che momento si riferiscono;
- stadio di conversione: i dataset sono stati convertiti da tabelle a RDF attraverso un sistema automatico che prevede poco intervento umano per la configurazione. L'intervento umano può essere utile per migliorare la qualità del risultato attraverso, ad esempio, la trasformazione di nomi di colonne in URI relativi a proprietà comuni.

I dati forniti dalle amministrazioni sono principalmente in formati tabellari. Per la loro trasformazione è stata individuata la corrispondenza tra colonne e proprietà, per cui ogni singola cella contiene il singolo valore della corrispondente proprietà per il record corrispondente. Proprietà e record sono automaticamente trasformati in URI, per cui il risultato finale prevede la costruzione di triple RDF individuate da: URI del record, Uri della proprietà e valore della cella.

Inoltre il sistema prevede la possibilità di collegare gli URI prodotti ad altri, anche esterni al portale, attraverso l'uso degli strumenti offerti dai sistemi per la creazione di ontologie, in particolare è stata usata la possibilità di identificare l'equivalenza tra URI in modo da fare riferimento agli stessi oggetti.

La visualizzazione e composizione dei risultati di interrogazioni è stato un altro aspetto importante nella realizzazione del portale di TWC. In particolare è stata creata una applicazione che compone query SPARQL su diversi dataset e le inoltra agli endpoint specifici. I risultati sono integrati nel caso presentino URI condivisi e sono convertiti in formati adatti ad essere visualizzati attraverso strumenti già disponibili allo scopo.

Questo modo di procedere ha dimostrato, secondo gli autori [15], che l'utilizzo di tecnologie Linked Data e de Web Semantico possono essere applicate per ridurre i costi di sviluppo, promuovere l'integrazione dei dati e aumentare il riuso di modelli di dati all'interno del dominio dell'amministrazione. Inoltre la facilità di sviluppo ha permesso la creazione di nuovi incroci di dati, dimostrando un basso costo per il trasferimento della conoscenza.

2.2 Analisi del problema

Alla luce dei lavori fin qui mostrati, il lavoro svolto in questo progetto è stato teso alla ricerca di un percorso per la creazione di un sistema che, a partire dall'utilizzo dei dati, già in possesso dei vari uffici comunali e relativi alla città di Bologna e ad i suoi abitanti, mettesse a disposizione tali dati secondo i principi propri del Web Semantico.

Come emerso dagli studi mostrati in precedenza, questo tipo di approccio nel fornire i dati permetterebbe di ridurre i costi, rispetto alla pubblicazione degli stessi all'interno di una relazione o un'applicazione. Questo, dal punto di vista degli utilizzatori dei dati, potrebbe comportare un problema di usabilità. Per superare questo problema, un requisito importante è stato quello di creare un sistema di supporto che permettesse di facilitarne l'utilizzo ai nuovi utenti e a quelli con scarse conoscenze tecniche dell'argomento.

Nel dettaglio, il progetto è partito dal recupero di dataset pubblicati, di norma come formato tabellare, dal comune di Bologna [9], dalla individuazione di dataset di interesse da poter utilizzare come esempio e su cui fosse possibile effettuare un confronto con altri dati disponibili alla luce di determinati casi d'uso di interesse.

A partire dai dati è stato deciso, allo stesso modo delle esperienze precedenti mostrate, che un obiettivo importante fosse quello di dare la possibilità di accedere ai dati, attraverso uno strumento standard di interrogazione del Web Semantico, anche a utenti con limitate conoscenze dello stesso. E inoltre, dal momento che i dati scelti sono in qualche modo collegati al territorio e ad una specifica zona, di aiutare la creazione di mappe che permettessero la visualizzazione di analisi statistiche su tali dati.

Il processo che si è sviluppato doveva tenere conto dalla possibilità di essere adattato a situazioni non specifiche ai dati del comune di Bologna, infatti altri dataset relativi ad altre amministrazioni comunali sono stati usati per validare il processo di sviluppo.

Infine un requisito tecnico importante è stato l'utilizzo di tecnologie open source nel processo di sviluppo per una specifica richiesta del committente, ovvero dell'amministrazione del comune di Bologna.

3 Background tecnologico

In questo capitolo verranno introdotti concetti più tecnici necessari per la comprensione del lavoro descritto in seguito. Tali dettagli tecnici sono comunque basati su quanto detto finora.
Di seguito verranno descritte più in dettaglio le tecnologie del web semantico e gli strumenti utilizzati per l'analisi dei dati. Successivamente verranno descritti gli strumenti utilizzati per conseguire i risultati richiesti dal progetto.
Con questo capitolo si completa il quadro delle conoscenze necessarie per poter continuare il percorso attraverso i successivi capitoli, in cui verrà data una descrizione del lavoro svolto.

3.1 Tecnologie del web semantico

Come detto in precedenza il Web è stato creato in origine per l'utilizzo da parte di persone. L'aggiunta di una semantica alle informazioni disponibili sul Web è in grado di rendere comprensibili ed utilizzabili tali informazioni anche per le macchine.
La visione semantica del Web è basata sui Linked Data, una organizzazione strutturata di informazioni statiche, che rende possibile collegare concetti diffusi su database differenti. Il Web Semantico fornisce diverse tecnologie per costruire quello che è chiamato il "*Web of Linked Data*". In questo capitolo ne verranno presentate solo la parte legata in modo più specifico al lavoro oggetto di questo documento.
Le tecnologie presentate riguardano:

- Resource Description Framework (RDF): viene usato per descrivere i linked data;
- RDF Schema (RDFS) e Web Ontology Language (OWL): vengono usati per definire un vocabolario di termini con lo scopo di esprimere una semantica per l'informazione;
- Protocollo SPARQL: è un linguaggio che viene usato per interrogare i linked data;

3.1.1 Resource Description Framework

Resource Description Framework (RDF) è una specifica del World Wide Web Consortium (W3C) rilasciata nel 1999. Progettato per l'elaborazione di metadati web, RDF fornisce una piattaforma di interoperabilità tra applicazioni sul web che si scambiano informazioni comprensibili dalle macchine. Il framework facilita l'elaborazione automatica delle informazioni e può essere utilizzato in diverse aree di applicazione: nella ricerca di risorse (*resource discovery*), per migliorare le capacità dei motori di ricerca, per descrivere il contenuto e le relazioni disponibili in un particolare sito web, pagina o digital library, per facilitare condivisione e scambio di conoscenza tramite gli agenti intelligenti, nella valutazione del contenuto, nella descrizione di collezioni di pagine che rappresentano un singolo documento *logico*, nella descrizione del copyright di pagine web, per esprimere le preferenze di un utente, per la gestione della privacy di un sito.

Rdf è basato su tre differenti tipi di dati:

- Uniform Resource Identifier (URI): sono riferimenti utilizzati per identificare le risorse come ad esempio http://www.w3.org/2000/01/rdf-schema#Class. La parte che segue il simbolo "#" è nota come *Fragment Identifier* (identificatore di frammento) [16]. Dal momento che gli URI possono condividere prefissi comuni, la notazione QName (Qualified Name) è spesso utilizzata per rendere più leggibile il documento e per brevità. Questa consiste nel definire un elemento prefisso che sostituisce un *namespace*, ovvero la parte radice dell'URI (la parte precedente al simbolo "#") e sostituita con una stringa, in genere più breve, seguita da ":". Nel caso dell'URI mostrato in precedenza la parte a sinistra del "#" può essere sostituita con "`rdfs:`". In questo modo lo stesso URI può essere ottenuto per sostituzione con "`rdfs:Class`" È anche possibile definire una stringa vuota per identificare un namespace. In quest'ultimo caso la notazione QName risulterebbe essere "`:Class`";
- Letterale (Literal): si tratta di una semplice stringa di caratteri e numeri delimitata da apici, rappresentate un valore concreto, e seguita da una opzionale specifica del tipo di dato contenuto, come ad esempio stringa, intero o data;

- Nodi vuoti (Blank Nodes): sono utilizzati per rappresentare concetti che non sono noti o non specificati. Possono definire risorse anonime che non sono identificate da un URI. Dal momento che non è specificato un URI per referenziare il concetto il prefisso è espresso tramite la notazione "_:". Un nodo vuoto non può essere usato per identificare globalmente una risorsa ma solo all'interno del documento RDF in cui è specificato.

3.1.1.1 Modello dei dati di un documento RDF

Il modello dai dati, ovvero la struttura logica dei dati, utilizzati da RDF è basato sull'idea di fare affermazioni (o *statement*) sui dati. Queste affermazioni sono espresse tutte nella forma "*soggetto predicato oggetto*", e per questo le affermazioni prendono il nome di *triple*. Una di queste triple rappresenta la più piccola unità di informazione presente in un documento RDF.

Le tre parti di uno statement RDF sono espressa da [2]:

- Risorse: tutte le cose descritte da espressioni RDF sono chiamate risorse. Una risorsa può essere un'intera pagina web, o una parte, o un particolare elemento all'interno di un documento HTML o XML. Una risorsa può anche essere una collezione di pagine come un intero sito web. O può essere un oggetto che non è direttamente accessibile via web come un libro. Le risorse sono sempre identificate tramite un URI (ogni cosa può avere un URI) o un nodo vuoto. Le risorse possono essere soggetto od oggetto di uno statement RDF.
- Proprietà: è uno specifico aspetto, caratteristica, attributo o relazione utilizzata per descrivere una risorsa. Ogni proprietà ha un significato specifico, definisce i valori permessi, i tipi di risorse che può descrivere e le relazioni con altre proprietà. Le proprietà sono sempre identificate come URI e costituiscono i predicati all'interno di uno statement RDF.

L'oggetto di una affermazione RDF può anche essere un dato di tipo letterale. Dal momento che il modello dei dati RDF è utilizzato per rappresentare i Linked Data ogni affermazione RDF può essere rappresentato come un grafo, in cui soggetto e oggetto costituiscono i nodi del grafo, mentre il predicato costituisce l'arco che li collega.

Proviamo a descrivere quanto detto in un esempio esplicativo. Prendiamo il caso di voler rappresentare il nome di una persona utilizzando RDF per descrivere le relazioni. Pr far utilizziamo il documento RDF mostrato in **Figura 3.1Figura 1.1**.

```
@prefix     vcard: <http://www.w3.org/2006/vcard/ns#>

<http://.../JohnSmith>        vcard:FN          "John Smith" ;
                              vcard:N           _:b1 .
_:b1                          vcard:Given       "John" ;
                              vcard:Family      "Smith" .
```

Figura 3.1: esempio di documento RDF

In questo esempio sono mostrati alcuni dei costrutti introdotti finora. In particolare è mostrato l'uso della notazione QName che sostituisce l'URI del predicato con la stringa vcard e unita al Fragment Identifier, ed anche il funzionamento dei nodi vuoti, in questo caso identificato dal simbolo "`_:b1`".
In **Figura 3.2** è mostrato il grafo risultante.

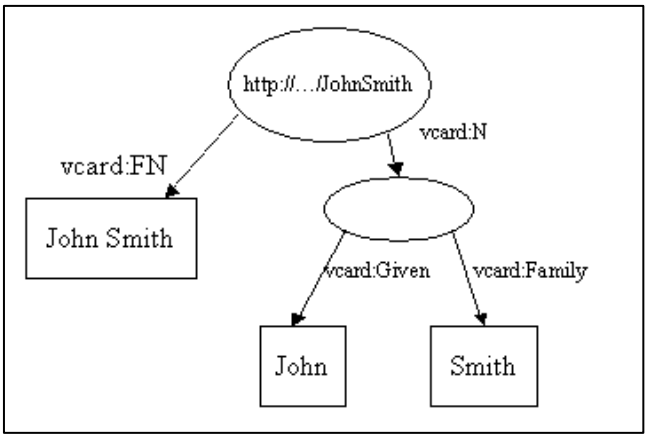

Figura 3.2: grafo del documento RDF mostrato in figura 2.1

RDF definisce un modello molto semplice per descrivere Linked Data che esprime relazioni tra le risorse. Il documento di esempio appena visto non è, tuttavia, l'unico modo per realizzarlo. Infatti esistono diversi formati per serializzare le informazioni di un grafo RDF.

3.1.1.2 Formati di serializzazione

Esistono diversi formati di serializzazione per produrre un documento descrittivo di uno specifico dataset a partire dal modello dei dati RDF. Di seguito verranno descritti i principali [17]:

- N-Triples: è il formato più semplice. Gli URI vengono espressi delimitandolo con parentesi angolari e le stringhe dei letterali sono delimitate da doppi apici. Ogni tripla occupa una singola riga e termina con un punto. La sua semplicità è quello che ha reso N-Triples molto popolare per insegnare alle persone ad usare RDF, e alcuni parser sono molto veloci ad analizzare questo formato perché hanno minor lavoro da svolgere, ma la verbosità del formato lo rende meno popolare di altri formati più compatti.
- RDF/XML (Resource Description Framework / Estensible Mark-Up Language): è il format più vecchio e fu parte della specifica originale di RDF nel 1999. In questo caso il grafo RDF è costruito utilizzando i tag del formato XML. I nodi del grafo possono essere costituiti sia da tag XML che da stringhe. Questo formato è il più prolisso e il più difficile da leggere e non è mai diventato molto popolare per varie ragioni, tra cui complicazioni nell'integrarsi con documenti XML o per la difficoltà con cui poteva essere processato da popolari strumenti per il formato XML.
- Turtle (Terse RDF Triple Language): è un format simile al N-Triples. Le differenze principali consistono nel poter definire un prefisso comune per gli URI e nella capacità di esprimere in un'unica riga più una tripla. In questo caso, però, le triple devono avere un soggetto comune, oppure soggetto e predicato comuni. In particolare:
    - notazione separata da virgola: unisce le triple che hanno in comune gli stessi soggetto e predicato;
    - notazione separata da punto e virgola: unisce le triple che hanno in comune lo stesso soggetto;
    - notazione QName: tramite la parola "`@prefix`" sostituisce un namespace URI con la specifica parola indicata.

Questo formato è sicuramente uno dei più usati per via della sua compattezza e facilità di lettura. Il documento mostrato in figura 2.1 risulta essere in formato Turtle;

- N3 (Notation 3): è un altro formato popolare per serializzare dati RDF. Fu un progetto personale di Tim Berners-Lee ed è caratterizzato da essere molto simile al formato Turtle. Infatti tutti i parser in gradi di analizzare N3 sono in grado di analizzare anche Turtle. Questo perché Turtle è una sottoparte di N3, infatti N3 offre in più alcune funzionalità e alcune scorciatoie per descrivere più fatti aventi lo stesso soggetto. In realtà queste funzionalità non sono molto usate ed N3 non è mai diventato uno standard.

```
<urn:isbn:006251587X> <http://purl.org/dc/elements/1.1/creator>
       <http://www.w3.org/People/Berners-Lee/card#i> .
<urn:isbn:006251587X> <http://purl.org/dc/elements/1.1/title> "Weaving the Web" .
```

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:v="http://www.w3.org/2006/vcard/">
  <rdf:Description rdf:about="urn:isbn:006251587X">
    <dc:title>Weaving the Web</dc:title>
    <dc:creator rdf:resource="http://www.w3.org/People/Berners-Lee/card#i"/>
  </rdf:Description>
</rdf:RDF>
```

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
<urn:isbn:006251587X> dc:creator <http://www.w3.org/People/Berners-Lee/card#i> ;
                      dc:title "Weaving the Web" .
```

Figura 3.3: formati di serializzazione RDF (in ordine N-Triples, RDF/XML e Turtle)

In **Figura 3.3** sono mostrati tre diversi formati di serializzazione (in ordine dall'alto N-Triples, RDF/XML e Turtle) in cui vengono descritti gli stessi concetti. Si nota subito che il formato RDF/XML risulta essere il più verboso, a differenza degli altri due, molto più compatti.

RDF è un linguaggio per descrivere Linked Data ma non specifica alcun meccanismo per descrivere relazioni tra risorse. Questa operazione è svolta da RDF Schema.

3.1.2 RDF Schema e OWL

RDF Schema (RDFS, noto anche come RDF vocabulary description language) è una estensione semantica per RDF introdotta con lo scopo di descrivere le ontologie. Fornisce un meccanismo di base per descrivere gruppi di risorse collegate tra loro e le relazioni tra le stesse risorse. È in grado di definire concetti e relazioni, e di definire vocabolari di termini per modellare specifici domini attraverso il linguaggio RDF stesso [18].

RDFS è basato su un set di classi e proprietà di default con cui è possibile specificare altre classi e proprietà, e specificare, così, un vocabolario completo. Il vocabolario costruito deve definire tutti i possibili concetti e tutte le relazioni tra i concetti, con lo scopo di poter costruire una struttura adatta a descrivere correttamente l'ambito informativo che si vuole modellare.

Il sistema di proprietà e classi di RDFS è simile a quello di un normale linguaggio ad oggetti, la differenza sta nel fatto che, invece di definire una classe in base alle proprietà che possono avere le istanze, descrive le proprietà in base alle classi di risorse a cui si applicano. Un beneficio di questo approccio focalizzato sulle proprietà è di permettere di estendere la descrizione di risorse esistenti molto più facilmente senza la necessità di ridefinire la descrizione originale, uno dei principi architetturali del web.

Per esempio, è possibile definire la proprietà "`eg:autore`" come avente un dominio di "`eg:Documento`" e un codominio di "`eg:Persona`"; mentre un comune linguaggio object oriented potrebbe definire tipicamente un classe "`eg:Libro`" con un attributo chiamato "`eg:autore`" di tipo "`eg:Persona`". Usando l'approccio di RDF risulta semplice per gli altri definire in seguito proprietà aggiuntive aventi come dominio "`eg:Documento`" e come codominio "`eg:Persona`". Questo può essere fatto senza la necessità di ridefinire la descrizione originale della classe.

3.1.2.1 Classi di RDFS

La parte principale del vocabolario RDFS è identificata dal namespace "`http://www.w3.org/2000/01 /rdf-schema#`" e nella notazione QName di solito

viene espressa tramite il prefisso "`rdfs:`", mentre la sintassi RDF è identificata dal namespace "`http://www.w3.org/1999/02 /22-rdf-syntax-ns#`" e di solito riferita col prefisso "`rdf:`".

Di seguito sono presentate le principali classi che compongono RDFS [18]:

- "`rdfs:Resource`": tutto ciò che è descritto da RDF è chiamato risorsa ed è, quindi una istanza di questa classe. Tutte le altre classi sono sottoclassi di questa classe. "`rdfs:Resource`" è una istanza di "`rdfs:Class`";
- "`rdfs:Class`": è la classe delle risorse RDF. E "`rdfs:Class`" è una istanza di se stessa;
- "`rdfs:Literal`": è la classe che identifica i letterali come stringhe e numeri. Valori delle proprietà come stringhe testuali sono esempi di questa classe. Possono essere tipati e in quel caso sono istanze della classe "`rdfs:Datatype`". Inoltre è una sottoclasse di "`rdfs:Resource`";
- "`rdfs:Datatype`": è una sottoclasse di "`rdfs:Literal`" ed è la classe che descrive i tipi di dato usati nel modello RDF;
- "`rdfs:XMLLiteral`": è la classe dei valori letterali XML. È una istanza di "`rdfs:Datatype`" e una sottoclasse di "`rdfs:Literal`";
- "`rdfs:Property`": è la classe delle proprietà RDF ed è un'istanza di "`rdfs:Class`".

Esistono anche risorse in grado di descrivere collezioni. La stessa risorsa può apparire più volte nello stesso contenitore e, diversamente dal contenimento del mondo fisico, un contenitore può contenere se stesso.

Esistono diversi contenitori che si distinguono dal modo di ordinare e selezionare gli oggetti contenuti.

3.1.2.2 Proprietà di RDFS

Di seguito sono presentate le principali proprietà che compongono RDFS [18]:

- "`rdfs:range`": è usata per indicare che i valori di una proprietà, oggetto di uno statement RDF, sono istanze di una o più classi specifiche. Ovvero per indicare il codominio di una proprietà. Possiamo dire, quindi che, se in uno statement RDF di esempio "S P O" (soggetto, predicato, oggetto),

definissimo “P rdfs:range C“ allora potremmo affermare che O deve essere una istanza di C;

- “`rdfs:domain`”: è utilizzata per indicare che ogni risorsa che ha una determinata proprietà deve essere una istanza della, o delle, classi specificate da questa proprietà. Per esempio se nello statement “S P O” si definisse che “P rdfs:domain C” allora S deve essere una istanza di C;
- “`rdf:type`”: questa proprietà è usata per descrivere l’appartenenza di una risorsa ad una classe. In sostanza definisce la relazione *instance of* (istanza di) tra il soggetto e l’oggetto. Al suo posto, all’interno di una tripla, può essere usato “`a`” come scorciatoia;
- “`rdfs:subClassOf`” e “`rdfs:subProperyOf`”: sono proprietà RDFS usate per indicare che una classe, o una proprietà, è sottoclasse, o sottoproprietà, di un’altra, rispettivamente, classe o proprietà. Per esempio, una istanza I della classe C1, sarebbe anche istanza di una classe C2 se C1 fosse sottoclasse di C2.

$$I\ rdf{:}type\ C1$$
$$C1\ rdfs{:}subClassOf\ C2$$
$$I\ rdf{:}type\ C2$$

- “`rdfs:label`”: è una proprietà che può essere utilizzata per fornire una versione leggibile per le persone del nome della risorsa;
- “`rdfs:comment`”: è una proprietà usata per fornire una descrizione della risorsa utile per le persone.

```
:Persona rdf:type rdfs:Class .
:moglieDi rdf:type rdfs:Class .
:coniuge rdf:type rdfs:Property .
:moglieDi rdfs:subPropertyOf :coniuge .
:Donna rdfs:subClassOf :Persona .
:Uomo rdfs:subClassOf :Persona .
:madreDi rdfs:domain :Donna .
:haFiglio rdfs:range :Uomo .
```

Figura 3.4: esempio di descrizione RDFS

In **Figura 3.4** è mostrato un esempio di descrizione RDFS in cui sono ricreati parte dei concetti per la rappresentazione di una genealogia.
In **Figura 3.5** ne è mostrato un ipotetico uso in cui si afferma che *Carmela sia una donna mentre Vito e Fredo degli uomini* e quindi *tutti e tre delle persone* in quanto *uomo e donna sono sottoclassi di persona*. *Carmela è moglie di Vito* e perciò è anche un coniuge dato che *moglieDi è una sottoporprietà di coniuge*. La proprietà *madreDi ha come dominio una Donna* per cui è corretto affermare che *Carmela è madre di Fredo*, per concludere la proprietà *haFiglio ha come codominio un Uomo*, per cui l'oggetto dell'ultima tripla, mostrata nell'esempio, deve essere necessariamente un uomo, come di fatto è.

```
ex:Carmela rdf:type :Donna .
ex:Vito rdf:type :Uomo .
ex:Fredo rdf:type :Uomo .
ex:Carmela :moglieDi ex:Vito .
ex:Carmela :madreDi ex:Fredo .
ex:Carmela :haFiglio ex:Fredo .
```

Figura 3.5: esempio di dataset basato sulla descrizione di **Figura 3.4**

Esistono molti schemi e ontologie che permettono di definire vincoli alla struttura dei dati ed è possibile definirne di nuovi secondo le esigenze. Tra gli esempi più citati ci sono sicuramente: il progetto *Friend of a Friend* (FOAF) [19] in cui si cerca di modellare una rete di relazioni tra persone e con le loro attività, oppure una trasformazione in RDF del formato vCard [20] [21]. Oppure un sistema più articolato che permette di specificare concetti che non fanno riferimento solo ad ambiti specifici, come ad esempio il Dublin Core Metadata Initiative [22].

3.1.2.3 Web Ontology Language (OWL)

RDF Schema è uno strumento che consente (in modo molto basilare) di definire vocabolari RDF, tuttavia in alcuni casi può essere utile utilizzare altri linguaggi che forniscono qualche capacità in più. Alcune di queste capacità sono state raccolte all'interno di Web Ontology Language [23]. Si tratta di un altro importante pezzo per la costruzione del Web Semantico.

OWL è una raccomandazione W3C con lo scopo di descrivere il significato semantico dei dati, presenta strumenti più espressivi di RDFS per esprimere le informazioni semantiche dei dati di cui ne costituisce una estensione.
La maggiore espressività, rispetto ad RDFS è dovuta alla capacità di descrivere:

- vincoli di cardinalità (es . una persona ha una sola madre e un solo padre);
- possibilità di indicare che due classi definite in schemi differenti rappresentano lo stesso concetto. Proprietà molto utile all'interno di un ambito distribuito come il Web, per collegare risorse definite in due schemi indipendenti. È il caso di "`owl:sameAs`";
- possibilità di indicare che due istanze, definite separatamente, rappresentano lo stesso concetto;
- possibilità di indicare che una proprietà è transitiva;
- la possibilità di definire una nuova classe come combinazione di più classi esistenti.

La definizione e la realizzazione di queste (e altre) capacità sono lo scopo del gruppo di lavoro sui linguaggi per la definizione di ontologie.

3.1.3 Interrogazione dei dati tramite SPARQL

SPARQL è un acronimo ricorsivo che sta per *SPARQL Protocol and RDF Query Language* [24]. Si tratta di un linguaggio dichiarativo pensato per interrogare informazioni memorizzate in formato Linked Data con lo scopo di raccogliere informazioni da basi di conoscenza distribuite sul web. SPARQL è parte integrante del progetto Web Semantico del W3C e, usando le parole di Tim Berners-Lee, "provare ad usare il Web Semantico senza SPARQL è come provare ad usare un database relazionale senza SQL".
SPARQL non è stato pensato per interrogare dati relazionali, ma per interrogare dati conformi al modello RDF.

3.1.3.1 SPARQL e SQL (Structured Query Language)

Sia SPARQL che SQL sono entrambi linguaggi dichiarativi usati per esprimere richieste ad un *query engine*.

SQL è usato per ricercare informazioni memorizzate in forma di tabelle all'interno di database relazionali. Tramite SQL è possibile selezionare, all'interno di una tabella, una specifica riga e il valore contenuto un una delle colonne della tabella.
In un ambiente distribuito ed eterogeneo come il web, dove è possibile che le stesse informazioni siano memorizzate in maniera differente, non sarebbe possibile utilizzare le stesse regole per estrarre le informazioni. Infatti non ci sono regole precise per definire una rappresentazione di una informazione [25].
Come detto, invece, SPARQL lavora usando dati in formato Linked Data, dove l'informazione è rappresentata in forma di grafi. Per poter gestire questa caratteristica, quindi, l'approccio risulta essere basato sul paradigma del *pattern-matching*. Attraverso l'individuazione delle corrispondenze tra il grafo su cui si sta eseguendo la query e i vincoli imposti dalla query stessa, vengono estratti i risultati e legati i valori selezionati alle variabili definite nella query.
Di seguito è mostrato un esempio tratto da [17] in cui è mostrato, in **Figura 3.6Errore. L'origine riferimento non è stata trovata.**, una semplice rubrica.

```
@prefix ab: <http://learningsparql.com/ns/addressbook#> .
@prefix d: <http://learningsparql.com/ns/data#> .

d:i0432 ab:firstName "Richard" .
d:i0432 ab:lastName "Mutt" .
d:i0432 ab:homeTel "(229) 276-5135" .
d:i0432 ab:email "richard49@hotmail.com" .

d:i9771 ab:firstName "Cindy" .
d:i9771 ab:lastName "Marshall" .
d:i9771 ab:homeTel "(245) 646-5488" .
d:i9771 ab:email "cindym@gmail.com" .

d:i8301 ab:firstName "Craig" .
d:i8301 ab:lastName "Ellis" .
d:i8301 ab:email "craigellis@yahoo.com" .
d:i8301 ab:email "c.ellis@usairwaysgroup.com" .
```

Figura 3.6: esempio di documento RDF preso da [17]

Se a partire dai dati in **Figura 3.6**,si volesse sapere qual è l'email di "Craig", come mostrato in **Figura 3.8Errore. L'origine riferimento non è stata trovata.**, sarebbe sufficiente indicare nel campo "`WHERE`" della query le relazioni che devono essere verificate per poter risalire al risultato che ci interessa.

In questo caso le persone della rubrica sono identificate univocamente da un numero, e a ciascuno di essi sono associate alcune proprietà. Nella query SPARQL, dove il punto interrogativo ad inizio parola identifica una variabile, in **Figura 3.7Errore. L'origine riferimento non è stata trovata.** possieda due proprietà di cui una ha il vincolo di essere legata ad un letterale specifico, ovvero il nome della persona di cui si vuol conoscere l'email. Nel campo "`SELECT`" è indicato quale informazione riportare in output.

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT ?craigEmail
WHERE
{
     ?person ab:firstName "Craig" .
     ?person ab:email ?craigEmail .
}
```

Figura 3.7: query SPARQL per ricavare l'email di Craig a partire dal grafo in **Figura 3.6**

In **Figura 3.8Errore. L'origine riferimento non è stata trovata.** è riportato il risultato della query in cui si vede che la corrispondenza con la proprietà "`ad:email`" comporta due risultati come risulta essere dal grafo di origine.

```
------------------------------
 craigEmail
==============================
 "c.ellis@usairwaysgroup.com"
 "craigellis@yahoo.com"
------------------------------
```

Figura 3.8: output della query mostrata in **Figura 3.7**

SPARQL è progettato per interrogare ed unire differenti sorgenti di dati sul web, per cui non ci sono problemi ad interrogare sistemi con sorgenti di dati distribuiti. Contrariamente a SQL, SPARQL e RDF si basano su un modello di dati standardizzato che permette di semplificare l'esplorazione ed estendere i modelli dei dati esistenti in un ottica di dati aperti.

3.1.3.2 Struttura di una query SPARQL

SPARQL fornisce diversi modi con cui interrogare una base di dati con lo scopo di eseguire diverse azioni o recuperare risultati in forme diverse [17]:

- *SELECT*: è la principale forma di query usata. Permette di richiedere dati ad una collezione di informazioni. Gli elaboratori di query tipicamente mostrano il risultato dell'interrogazione con SELECT sotto forma di tabella dove è associata ad una colonna tutte o un sottoinsieme delle variabili utilizzate per individuare i risultati;
- *CONSTRUCT*: ha come output delle triple. Con questo tipo di query si possono estrarre triple dalle sorgenti dei dati senza cambiarle, oppure costruire nuove triple a partire dai dati estratti. Questo permette di copiare creare e convertire dati RDF;
- *ASK*: chiede all'elaboratore di query se lo schema del grafo contenuto nella query descrive o no un insieme di triple all'interno del particolare database. Questo tipo di query sono utili per esprimere regole su condizioni che devono o non devono essere verificate nei dati. Queste regole possono essere utilizzate per automatizzare il controllo della qualità e del processo di analisi dei dati;
- *DESCRIBE*: con questa query si richiede di descrivere una particolare risorsa. L'elaboratore della query determina autonomamente, come descritto nelle specifiche di SPARQL, quali triple utilizzare come risposta scelte tra quelle che hanno la risorsa come soggetto. Le risposte risultano essere inconsistenti tra le varie implementazioni, cosa che rende questo tipo di query poco popolare.

Tutte e quattro le forme di query, come detto, sono in grado di compiere il loro lavoro individuando una corrispondenza tra grafi, anche se presentano i risultati in formati diversi. Di seguito è trattata più approfonditamente sola la forma SELECT anche perché le altre forme risultano essere più rare.

Una query SPARQL è caratterizzata da una struttura composta da diverse parti [17] [24]:

- *le dichiarazioni dei prefissi*: per abbreviare gli URI delle risorse e per applicare la notazione QName, prima della vera e propria query, viene posta, opzionalmente, la parola chiave "`PREFIX`" seguita dalla stringa che sostituisce l'URI e URI stesso;

- *la clausola di indicazione dei risultati*: di seguito alla parola chiave "`SELECT`" vengono indicate le variabili utilizzate nella query di cui si vuole conoscere il risultato;
- *la definizione del dataset*: tramite le parole chiave "`FROM`" e "`FROM NAMED`" seguite da un URI è possibile specificare uno o più grafi RDF su cui eseguire la query;
- *lo schema della query*: identificata dalla parola chiave "`WHERE`" è presente la parte centrale dalla query. In questa area è possibile specificare uno schema di relazioni che le triple del grafo RDF, che si sta interrogando, devono soddisfare per poter essere selezionate per l'output. È identificato anche come *graph pattern* ed è espresso in formato Turtle;
- *i modificatori del risultato*: l'ultima parte della query prevede degli operatori opzionali, ed usabili in concerto, per poter riorganizzare i risultati ottenuti:
    - *LIMIT e OFFSET*: queste parole chiave seguite da un valore numerico permettono di limitare il numero dei risultati da presentare in output nel primo caso, o, nel secondo caso, di scartare i primi risultati;
    - *ORDER BY*: permette di ordinare i risultati in modo crescente o discendente per il valore di una delle variabili della soluzione;
    - *GROUP BY*: permette di aggregare i risultati secondo i valori della variabile specificata. Di conseguenza è necessario definire come aggregare i risultati delle altre variabili portate in soluzione. Inoltre l'aggregazione permette di eseguire altre operazioni sui risultati sui raggruppamenti dei risultati come operazioni sui valori numerici delle variabili (calcolo di valori medio, minimo e massimo), oppure il conteggio delle occorrenze.

In **Figura 3.9** è mostrata una query di esempio in cui sono mostrate alcune delle cose descritte fin qui. All'interno del grafo RDF "`<http://my_expenses/meal>`" vengono selezionati gli elementi che contengono le proprietà "`e:descritpion`" e "`e:amount`". I risultati vengono aggregati secondo i valori della variabile "`?description`" e viene calcolata la somma dei valori di "`?amount`" per cui il valore corrispondente di "`?description`" è identico.

```
PREFIX e: <http://example.org/ns/expenses#>

SELECT ?description (SUM(?amount) AS ?mealTotal)
FROM <http://my_expenses/meal>
WHERE
{
     ?meal e:description ?description ;
           e:amount ?amount .
}
GROUP BY ?description
```

Figura 3.9: semplice query SPARQL di esempio

Questo esempio è molto semplice ed è formato da solo due triple, è possibile specificare diversi tipi di schemi più complessi anche combinando tra loro semplici schemi.

3.1.3.3 Schemi di query SPARQL

Tutti i graph pattern sono espressi usando il formato Turtle di RDF. Le variabili sono indicate dal simbolo"?" e possono essere usate per sostituire qualsiasi elemento di uno statement RDF; anche tutti gli elementi di una tripla possono essere composti da variabili.

I possibili tipi di graph pattern che sono usati in query SPARQL sono [24]:

- *Basic Graph Pattern*: è formato da un insieme di triple RDF che devono essere considerate insieme. In questo modo è possibile selezionare specifici rami del grafo RDF. Dopo che è stato trovato un riscontro tra la struttura della query e del grafo RDF interrogato, le variabili sono legate ai valori specifici trovati producendo la soluzione;
- *Group Graph Pattern*: è un insieme di uno o più graph pattern delimitati da parentesi graffe. Tutti i pattern devono essere avere un riscontro sulla stessa parte di grafo per far si che possa essere selezionata per la soluzione;
- *Optional Graph Pattern*: è un insieme di graph pattern opzionali che possono o non possono fare match. È indicato dalla parola chiave OPTIONAL ed nel caso non dovesse verificarsi riscontro la soluzione resta invariata lasciando semplicemente le variabili non legate ad uno specifico valore;
- *Alternative Graph Pattern*: è costituito da un insieme di due o più graph pattern dove vengono selezionate le soluzioni per ogni pattern preso

singolarmente, e dove il risultato è costituito unendo i singoli risultati. Questo graph pattern è identificato dalla parola chiave UNION inserita tra i singoli pattern;

- *Patterns on Named Graphs*: si tratta di un graph pattern che viene eseguito su di uno specifico grafo RDF identificato da un URI. Inoltre il nome del grafo può essere a sua volta identificato tramite una variabile e quindi essere parte della soluzione cercata. Questo tipo di pattern è identificato dalla parola chiave GRAPH.

Altri due elementi fondamentali di una query SPARQL sono la clausola per filtrare i risultati e le sotto-query. La clausola di filtratura, identificata dalla parola chiave FILTER, è usata per ridurre il numero dei risultati sulla base di vincoli specifici. Le sotto-query, invece, non sono altro che query inserite all'interno di altre per annidare i risultati di una query in un'altra.

```
@prefix : <http://people.example/> .

:alice      :name       "Alice",
                        "Alice Foo",
                        "A. Foo" .
:alice      :knows      :bob,
                        :carol .
:bob        :name       "Bob",
                        "Bob Bar",
                        "B. Bar" .
:carol      :name       "Carol",
                        "Carol Baz",
                        "C. Baz" .
```

Figura 3.10: documento RDF di esempio

Prendiamo in esame il documento RDF mostrato in **Figura 3.10** che definisce una serie di relazioni tra persone ovvero il nome (o i nomi) con cui è indicato un singolo individuo e i suoi rapporti di conoscenza. Volendo calcolare il nome più corto usato per indicare le persone che conosce Alice dobbiamo eseguire la query mostrata in **Figura 3.11** che è costituita da una query contenuta in un'altra.

La sotto-query viene eseguita per prima e seleziona soltanto il primo dei nomi ordinati in modo alfabetico di ogni persona. Questi risultati devono essere confrontati con il pattern della query esterna e verificata la corrispondenza.

```
PREFIX : <http://people.example/>

SELECT ?y ?minName
WHERE {
       :alice :knows ?y .
       {
              SELECT ?y (MIN(?name) AS ?minName)
              WHERE {
                     ?y :name ?name .
              } GROUP BY ?y
       }
}
```

Figura 3.11: query da eseguire su dataset di esempio in **Figura 3.10Figura 3.10**

Il risultato finale dell'elaborazione, mostrato in **Figura 3.12**, che nel passaggio intermedio dell'elaborazione alla fine della sotto-query conteneva una riga in più relativa ad "`:alice`"

```
----------------------
Y          |minName
======================
:bob       |"B. Bar"
:carol     |"C. Baz"
----------------------
```

Figura 3.12: risultato delle query in **Figura 3.11**

3.2 Trasformazione dei dati e visualizzazione dei risultati

Di seguito sono mostrati i principali strumenti utilizzati all'interno di questo progetto per raggiungere gli obiettivi preposti. Essendo, la maggior parte del lavoro, stato focalizzato su produzione ed interrogazione di grafi in formato RDF, la maggior parte degli strumenti utilizzati sono utili proprio in questo ambito. Infine è mostrato uno strumento GIS che è stato utilizzato per visualizzare su mappa i risultati delle elaborazioni su dati legati ad aree specifiche.

3.2.1 Apache Any23

Anything To Triples (Any23) è una libreria integrabile in altre applicazioni, un web service e uno strumento da linea di comando che trasforma in formato RDF dati strutturati memorizzati in documenti di vari formati. Supporta diversi formati di input, come la possibilità di estrarre informazioni semantiche incluse in una pagina

web (è il caso di RDFa [26], un particolare formato RDF che permette di mischiare testo e informazioni semantiche all'interno di un'unica pagina web) o da file CSV [12]. Apache Any23 è scritto in Java ed è pubblicato sotto licenza Apache v2.0.

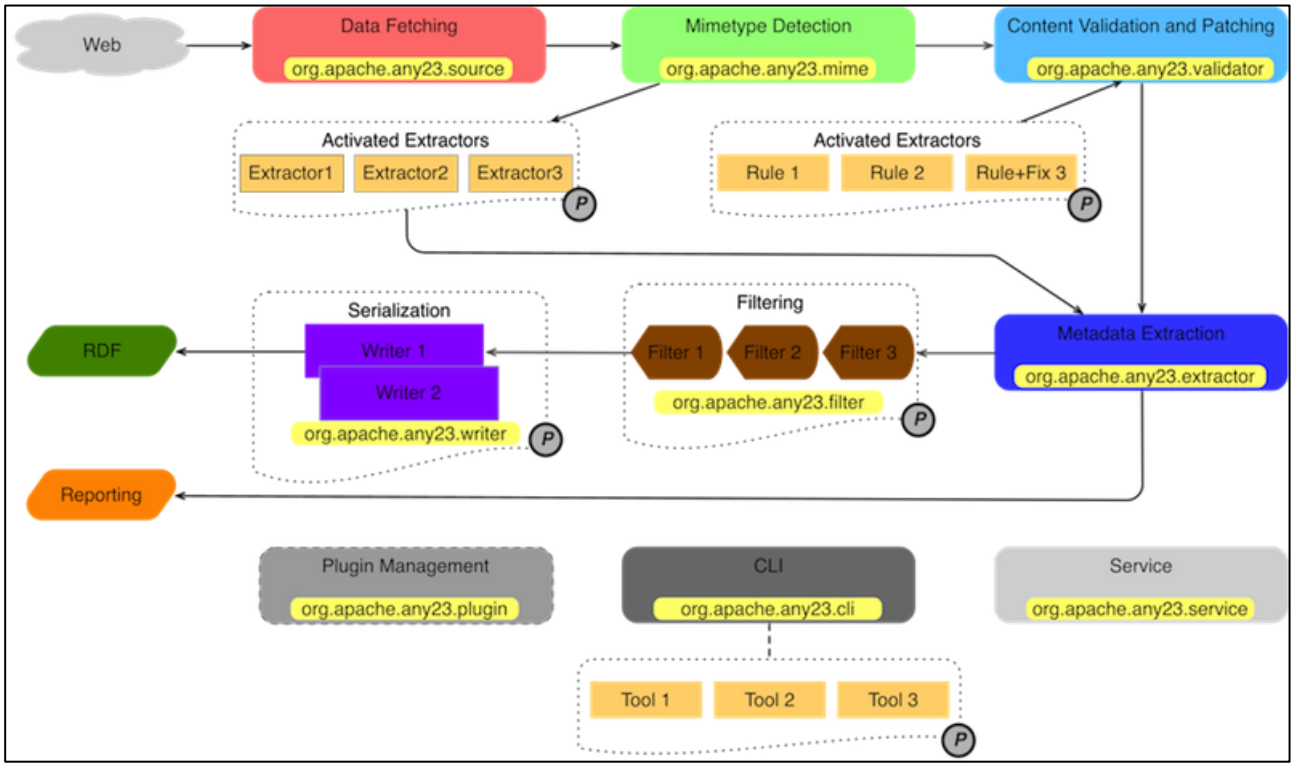

Figura 3.13: moduli logici di Apache Any23, flusso dei dati e package che implementano le funzionalità

Any23 è composto da vari moduli logici che, insieme, permettono il recupero e la trasformazione dei dati [27]:

- Data Fetching: questo modulo è responsabile di recuperare i dati grezzi dal Web. I dati collezionati da questo modulo analizzati dal modulo MIMEtype Detection;
- MIMEtype Detection: questo modulo permette di individuare il formato con cui sono codificati i dati e il contenuto del tipo MIME (standard di internet per indicare il formato dei file inviati tramite protocolli HTTP o SMTP [28]). Questa identificazione viene, poi, usata per attivare il modulo adatto per permettere l'estrazione dei dati;
- Content Validation and Patching: questo modulo si occupa di individuare e, nella maggior parte dei casi, di correggere problemi ed errori minori presenti nei dati esposti su pagine Web. L'individuazione e la correzione sono basate

su di un set di regole estendibile. Questo modulo si applica sono per documenti basati su DOM (modello a oggetti del documento [29]), e quindi documenti HTML;

- Metadata Extraction: questo modulo provvede ad eseguire tutti i moduli di estrazione selezionati durante le fasi di analisi precedenti e genera gli statement RDF insieme ad un rapporto sugli errori;
- Metadata Filtering: i dati in uscita dal modulo precedente possono essere filtrati per rimuovere dati spuri, ripetuti o non desiderati, tramite questo modulo;
- Serialization: infine le triple RDF ottenute sono convertite in uno dei formati per la serializzazione di RDF.

Esistono altri moduli che svolgono funzioni ausiliarie all'interno dell'applicazione. Ad esempio è disponibile un modulo che permette l'estensione della piattaforma attraverso l'individuazione e registrazione a runtime di componenti addizionali, ad esempio per permettere di estendere il numero di formati supportati sia in input che in output.

In **Figura 3.13** è descritta l'architettura di Apache Any23, in figura è evidenziato il percorso logico compiuto dai dati durate la fase di elaborazione e trasformazione degli stessi.

Il modulo di estrazione per i file CSV produce una rappresentazione in formato RDF conforme con la definizione del formato CSV. L'estrattore prevede la presenza di una intestazione per le colonne del file CSV, e tali intestazioni sono usate per identificare i predicati all'interno degli statement RDF prodotti.

La costruzione del grafo RDF a partire da un file CSV prevede la possibilità di indicare un indirizzo URI per indicare le risorse che stanno per essere create. Se le intestazioni delle colonne sono, a loro volta, già degli URI allora vengono utilizzati senza modifiche per indicare i predicati all'interno del grafo affinché siano dereferenziabili. Altrimenti, se le intestazioni non sono degli URI validi, il nome viene concatenato con l'URI associato al file CSV.

Inoltre, vengono aggiunte una serie di altre triple che permettono di ricostruire il file CSV a partire dal RDF di output. Viene aggiunta una tripla per ogni intestazione di colonna per associare, con predicato "`rdfs:label`", l'URI prodotto dalle rispettive

intestazioni col nome originale, e un altro statement per indicare il numero d'ordine della rispettiva colonna. Ad ogni riga è associato un numero d'ordine e tanti predicati quante sono le proprietà individuate a partire dalle intestazioni delle colonne e, con oggetto di tali statement, il valore della cella corrispondente.

```
Provincia_Comune;Comune;COD_QUAR;QUAR_NOME;COD_ZONA;ZONA_NOME
37006;Bologna;1;Borgo Panigale;C;Borgo Panigale
37006;Bologna;2;Navile;B;Bolognina
37006;Bologna;2;Navile;E;Corticella
37006;Bologna;2;Navile;I;Lame
37006;Bologna;3;Porto;M;Marconi
37006;Bologna;3;Porto;P;Saffi
```

Figura 3.14: esempio di CSV in input

```
<http://example.com#row/0> a <http://vocab.sindice.net/csv/Row> ;
     <http://example.com#Provincia_comune>
          "37006"^^<http://www.w3.org/2001/XMLSchema#integer> ;
     <http://example.com#Comune>
          "Bologna"^^<http://www.w3.org/2001/XMLSchema#string> ;
     <http://example.com#Cod_quar>
          "1"^^<http://www.w3.org/2001/XMLSchema#integer> ;
     <http://example.com#Quar_nome>
          "Borgo Panigale"^^<http://www.w3.org/2001/XMLSchema#string> ;
     <http://example.com#Cod_zona>
          "C"^^<http://www.w3.org/2001/XMLSchema#string> ;
     <http://example.com#Zona_nome> "Borgo
          Panigale"^^<http://www.w3.org/2001/XMLSchema#string> .
```

Figura 3.15: parte di output dell'elaborazione con Any23 di input in **Figura 3.14**

In **Figura 3.14** è mostrato un file CSV di esempio in cui le colonne sono delimitate da punto e virgola e la prima riga mostra i nomi delle rispettive colonne.
In **Figura 3.15** è mostrato l'output relativo alla prima riga, di dati, del file mostrato in **Figura 3.14**. In cui si vede chiaramente che i predicati sono costruiti a partire dal nome dato alle colonne, e come oggetto è inserito il contenuto della corrispondente cella.

3.2.2 OpenLink Virtuoso Opensource

OpenLink Virtuoso Opensource è un database engine, pubblicato con licenza GNU General Public License versione 2, che combina diverse funzionalità per la gestione di basi di dati di natura diversa. In particolare è in grado di gestire dati memorizzati secondo il modello tradizionale dei database relazionali ed anche database relazionali

ad oggetti e database federati; o anche database in formati XML o RDF oltre ad essere in grado di offrire funzionalità di Web Service. Il tutto tramite un'architettura ibrida che, invece di attivare per ogni funzionalità disponibile un server dedicato, attiva un singolo processo server multithread che implementa i vari protocolli disponibili. La versione commerciale di questo prodotto è anche nota come Virtuoso Universal Server proprio per la sua capacità di integrare diversi modelli dei dati e di integrare facilmente i vari modelli.

All'interno di questo progetto, Openlink Virtuoso Opensource è stato usato unicamente per gestire i dati in formato RDF e per accedere agli stessi attraverso l'endpoint SPARQL messo a disposizione dal sistema.

L'implementazione per il supporto RDF e SPARQL è costruita estendendo la già esistente infrastruttura del database relazionale ad oggetti [30], in modo da riutilizzare buona parte delle funzionalità preesistenti. L'implementazione SPARQL è stata costruita a partire dalla struttura di base dell'implementazione SQL. Le query SPARQL sono convertite in query SQL che vengono, così, eseguite sul database relazionale interno e, infine, il risultato viene trasformato in formato RDF.

Virtuoso offre una interfaccia di gestione, per poter inserire e gestire i grafi prodotti, sia da browser che da shell, inoltre è possibile interrogare il sistema sia tramite una interfaccia Web o inoltrare la richiesta attraverso un'applicazione remota tramite protocollo HTTP.

3.2.3 Apache Jena

Apache Jena è un framework che fornisce una collezione di strumenti e librerie Java con lo scopo di aiutare lo sviluppo di applicazioni e strumenti utili per operare con Linked Data e Web Semantico e pubblicato sotto licenza Apache v2.0.

Questo framework include strumenti per elaborare e scrivere dati RDF nei diversi formati di serializzazione mostrati nel paragrafo 3.1.1.2, per la gestione di ontologie e per effettuare *reasoning* con sorgenti di dati OWL e RDFS, per memorizzare in locale triple RDF e un motore per query SPARQL per effettuare le query sui dati.

Come mostrato in **Figura 3.16** l'architettura di Jena è molto articolata e presenta diverse funzionalità. In particolare noi siamo interessati ad alcune API SPARQL e alla possibilità di gestire il risultato di una query.

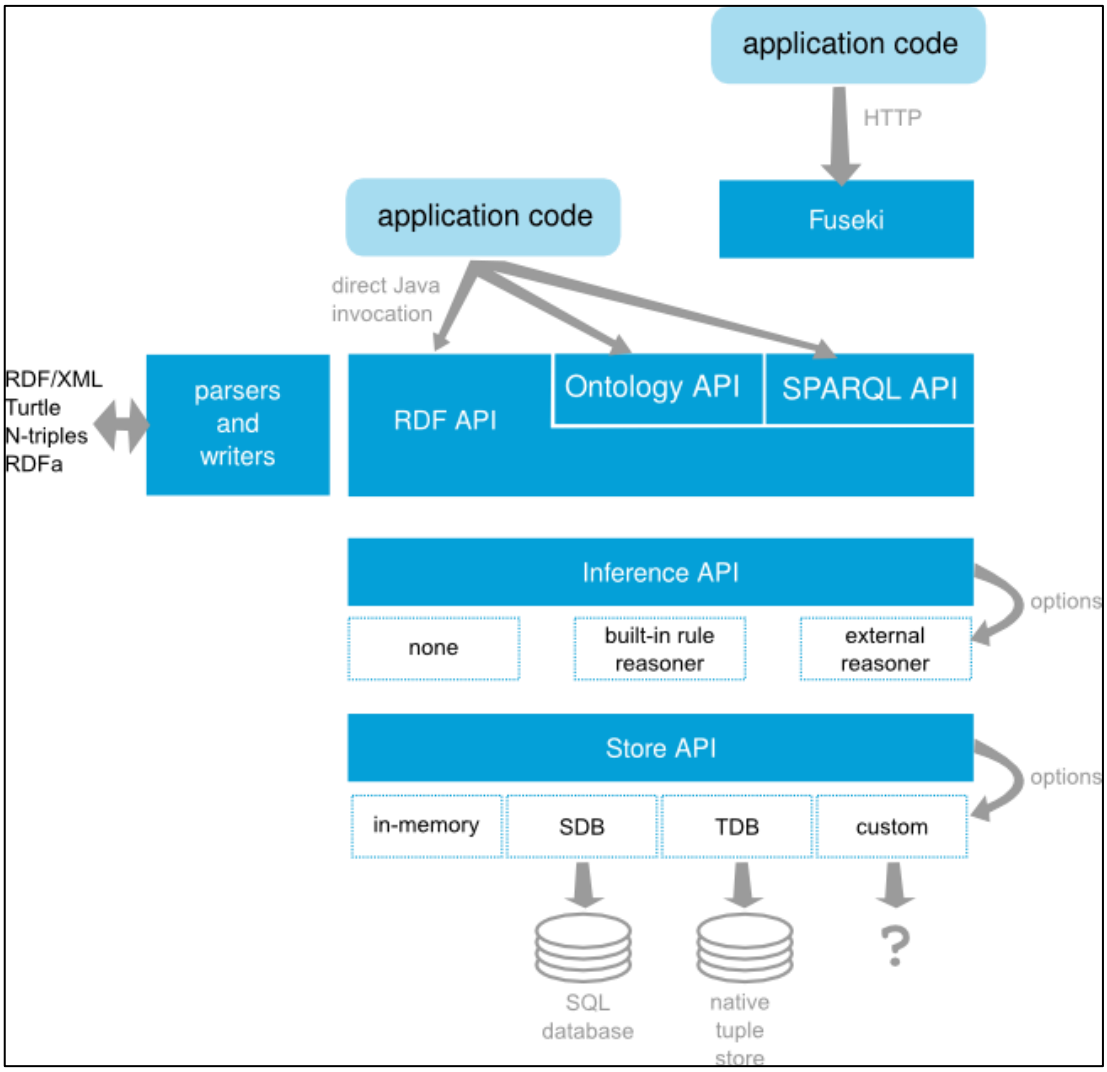

Figura 3.16: architettura di Apache Jena

In particolare le classi di interesse sono [31]:

- "`Query`": una classe che rappresenta la query dell'applicazione. Si tratta di un contenitore per tutti i dettagli della query. Gli oggetti di classe `Query` sono creati attraverso la chiamata di uno dei metodi di "`QueryFactory`" che forniscono accesso a diversi decodificatori per creare la query da stringa o da file di testo;
- "`QueryExecution`": questa classe rappresenta l'esecuzione di una query su cui può essere invocata l'esecuzione della stessa;
- "`QueryExecutionFactory`": classe usata per ottenere una istanza di `QueryExecution` a partire da un oggetto `Query` e un riferimento al servizio endpoint SPARQL;

- Nel caso di una query SELECT:
    - "`QuerySolution`": un oggetto che contiene le soluzioni della query e che li mostra singolarmente;
    - "`ResultSet`": un iteratore su tutte le risposte alla query;
    - "`ResultSetFormatter`": una classe in grado di trasformare un `ResultSet` in vari modi tra cui semplice testo, in formato CSV o in un grafo RDF (serializzabile poi nei vari formati supportati).

Per le altre tipologie di query SPARQL la differenza principale consiste nel formato delle risposte. Ad esempio una query ASK prevede solo un valore booleano come output.

All'interno di questo progetto il framework Jena è stato usato per creare un'applicazione per interrogare l'endpoint SPARQL gestito dal server Virtuoso, visto in precedenza.

3.2.4 QuantumGIS

QuantumGIS (noto anche come QGIS) è un sistema informativo territoriale opensource, pubblicato con licenza GNU General Public License, multipiattaforma che offre differenti applicazioni per le varie esigenze [32]:

- QGIS desktop: applicazione classica che offre strumenti per visualizzare, modificare e analizzare dati cartografici;
- QGIS Browser: visualizzatore di dati cartografici disponibili in locale o presso server remoti;
- QGIS Server: un server WMS (Web Map Service) in grado di realizzare un servizio di fornitura remota di mappe in formati raster;
- QGIS Client: una interfaccia web per la visualizzazione di dati cartografici.

3.2.4.1 QGIS Desktop

Le principali caratteristiche di QGIS Desktop comprendono [32]:

- la possibilità di visualizzare immagini cartografiche sia raster che vettoriali, sia disponibili in locale che attraverso l'interrogazione di servizi remoti preposti alla fornitura di dati cartografici;

- la possibilità di modificare, creare ed esportare in vari formati oltre che la possibilità modificare la proiezione utilizzata dalle mappe, e quindi riproiettare le mappe al volo;
- permettere l'analisi spaziale dei dati attraverso strumenti forniti dall'ambiente stesso;
- la possibilità di estendere le funzionalità di questo strumento attraverso il supporto ad una architettura a plugin scritti in linguaggio Python [33].

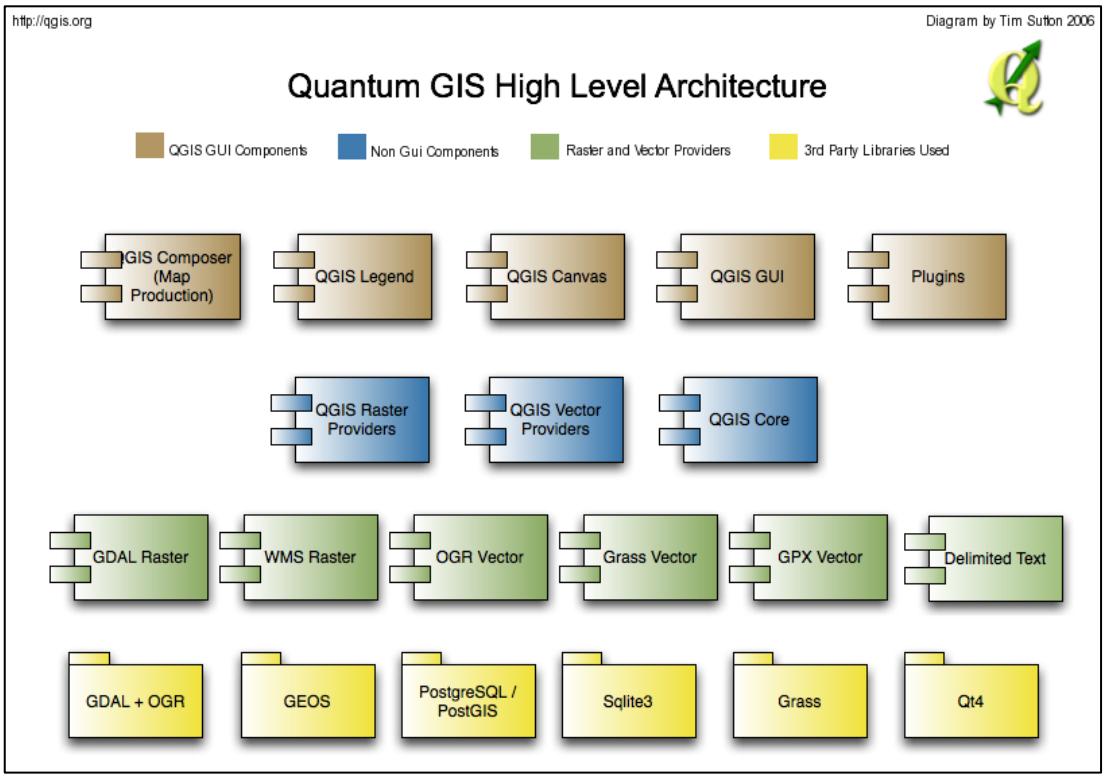

Figura 3.17: visione architetturale ad alto livello di QuantumGIS

In **Figura 3.17** viene presentata la divisione in componenti funzionali che caratterizza QuantumGIS. Sono presenti componenti in grado di gestire sia dati in formato raster che vettoriale. I componenti di supporto per l'esecuzione dei plugin permette di accedere all'area di lavoro e permettere operazioni di visualizzazione, oltre ad essere in grado di compiere operazioni sui layer nell'area di lavoro. La libreria Qt4 [34] è utilizzata per la costruzione dell'interfaccia e la visualizzazione dei risultati.

Dal momento che QGIS è sviluppato in linguaggio C++ per il supporto ai plugin in Python possiede internamente uno strato di collegamento per permettere

correttamente l'interazione tra i plugin e le librerie interne. Per la creazione dell'interfaccia il sistema fa uso delle librerie Qt4 e delle librerie di collegamento PyQt sviluppate dall'azienda Riverbank [35] per il linguaggio Python. Qt, come il resto dei componenti principali di QuantumGIS, è sviluppata in C++. La libreria PyQt, quindi, permette di invocare le API di Qt4

Inoltre è prevista la possibilità di pubblicare, tramite repository, i plugin prodotti e di accedere a quelli prodotti da altri.

I dati caricati vengono visualizzati come layer, ovvero come strati che possono comporre una figura più particolareggiata. Ad esempio una layer sui confini di una città caricato insieme al layer delle strade e al layer dei numeri civici o degli edifici assieme ad ortofoto della zona di interesse. Questo permette di creare visualizzazioni specifiche o di operare elaborazioni solo su parti di dati.

La libreria delle API messe a disposizione da QGIS permette di accedere ai dati dei singoli layer caricati e consentono di operare sia sui dati geometrici che sulle informazioni tabellari associate alle singole geometrie, nonché alle modalità di visualizzazione delle stesse e di permettere l'aggiunta di etichette e grafici sulla mappa.

4 Sistema di gestione ed estrazione dei dati

Nei precedenti capitoli sono state mostrate le capacità e le possibilità offerte dagli strumenti sviluppati all'interno del progetto del Web Semantico. Sono stati mostrati, inoltre, esempi ed esperienze nell'ambito della realizzazione di sistemi e metodologie per la pubblicazione di dati, affinché fossero coerenti con i principi del Web Semantico.

In questo capitolo, dopo un primo discorso introduttivo sulle parti del progetto e una esposizione di base sulla struttura dell'architettura, verranno mostrate le fasi realizzative che hanno portato alla creazione di un percorso completo del trattamento dei dati, a partire dai dati rilasciati dalle amministrazioni pubbliche sui rispettivi portali, fino alla realizzazione di rappresentazioni grafiche dei risultati di interrogazioni su tali dati.

4.1 Architettura della soluzione

Il problema di fornire un accesso ai dati delle pubbliche amministrazioni attraverso strumenti che permettessero una facile interrogazione dei dati stessi, e visualizzazione dei risultati, è stato affrontato, e portato a termine, attraverso una soluzione che individua sia un processo di elaborazione dei dati, sia attraverso la creazione di strumenti atti a facilitare il compito dell'utente finale dei dati.

Per quanto riguarda il processo di elaborazione, le fasi che lo costituiscono prevedono, dopo aver individuato il dataset da trasformare, prima di tutto una verifica della struttura del file in formato tabellare. Il vincolo principale in questa operazione, dovuto principalmente al comportamento dello strumento utilizzato per la trasformazione, è dovuto al fatto che, il file contenete la tabella con le informazioni del dataset, deve avere un record per riga e ogni colonna deve essere etichettata.

Una volta fatto ciò, il risultato dell'operazione può essere inserito all'interno del servizio web in grado di accettare richieste di interrogazione.

A questo punto, è necessario individuare dei casi d'uso per gli specifici dataset selezionati, a partire dai quali costruire le query in grado di fornire i dati per la fase di elaborazione successiva. Questa fase risulta essere centrale all'interno dell'elaborazione, in quanto la definizione delle interrogazioni permette di recuperare e ricombinare i dati, che sono memorizzati in maniera disgiunta.

Nel caso i dati elaborati siano in un qualche modo legati a specifiche zone del territorio comunale, o risultati statistici legati ad aree specifiche del territorio comunale o perché i risultati sono coordinate estrapolate da altri dati testuali, possono essere visualizzati come oggetti vettoriali sotto forma di grafico, su cartine all'interno di strumenti di visualizzazione di mappe cartografiche. Per far ciò è stata creata una estensione per l'ambiente GIS per facilitare la visualizzazione dei risultati.

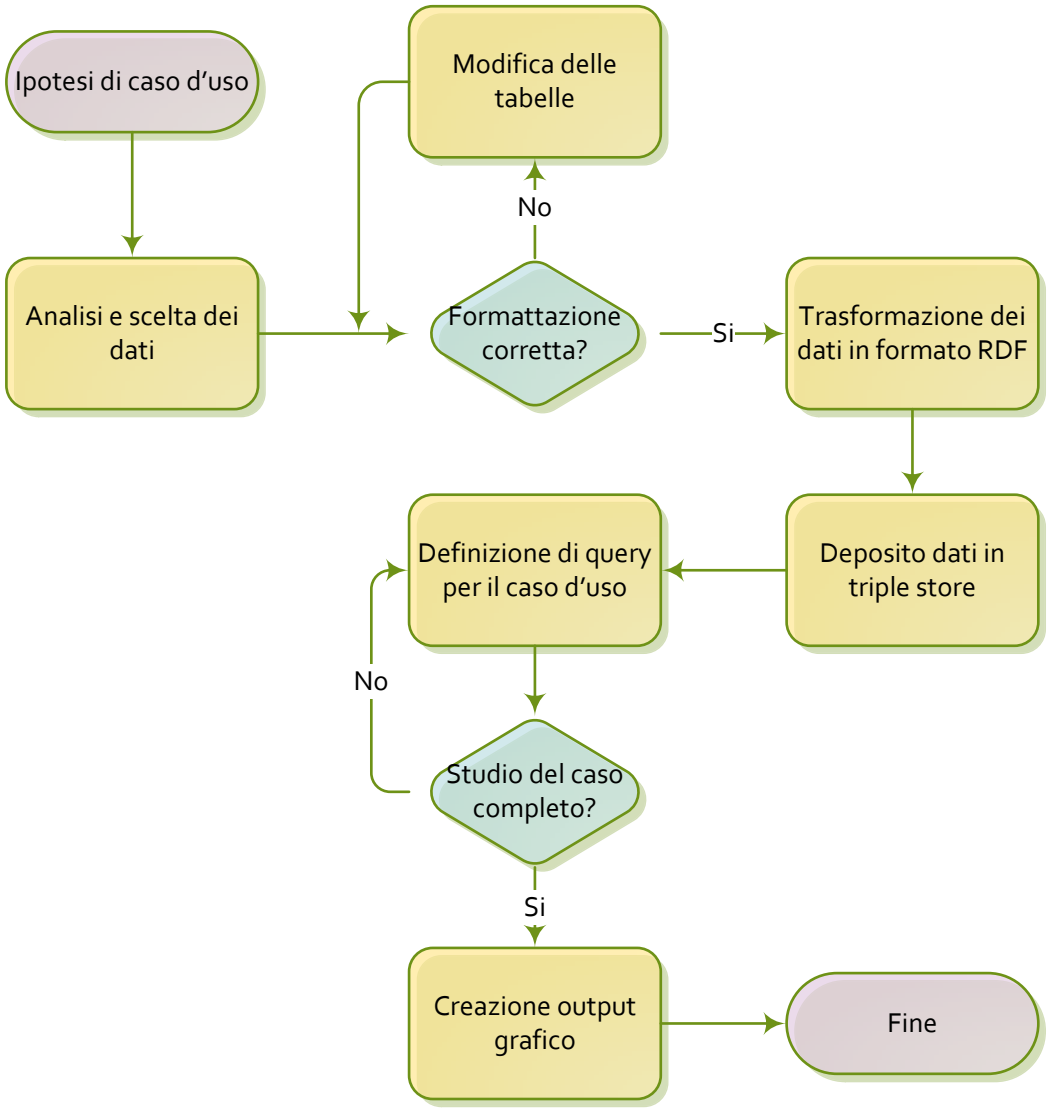

Figura 4.1: diagramma di flusso del processo realizzativo

In **Figura 4.1** è mostrato un diagramma di flusso che esemplifica quanto detto. Per poter creare una visualizzazione di interesse è necessario patire da una ipotesi sensata di utilizzo dei dati e incrocio degli stessi. In realtà, l'intera operazione può essere separata in due parti dal momento che i dati sono indipendenti dal loro utilizzo. Infatti sarebbe auspicabile la completa trasformazione e messa a disposizione dei dati in formato RDF per permettere la creazione di incroci di dati, anche con dati esterni, da parte degli stessi utenti.

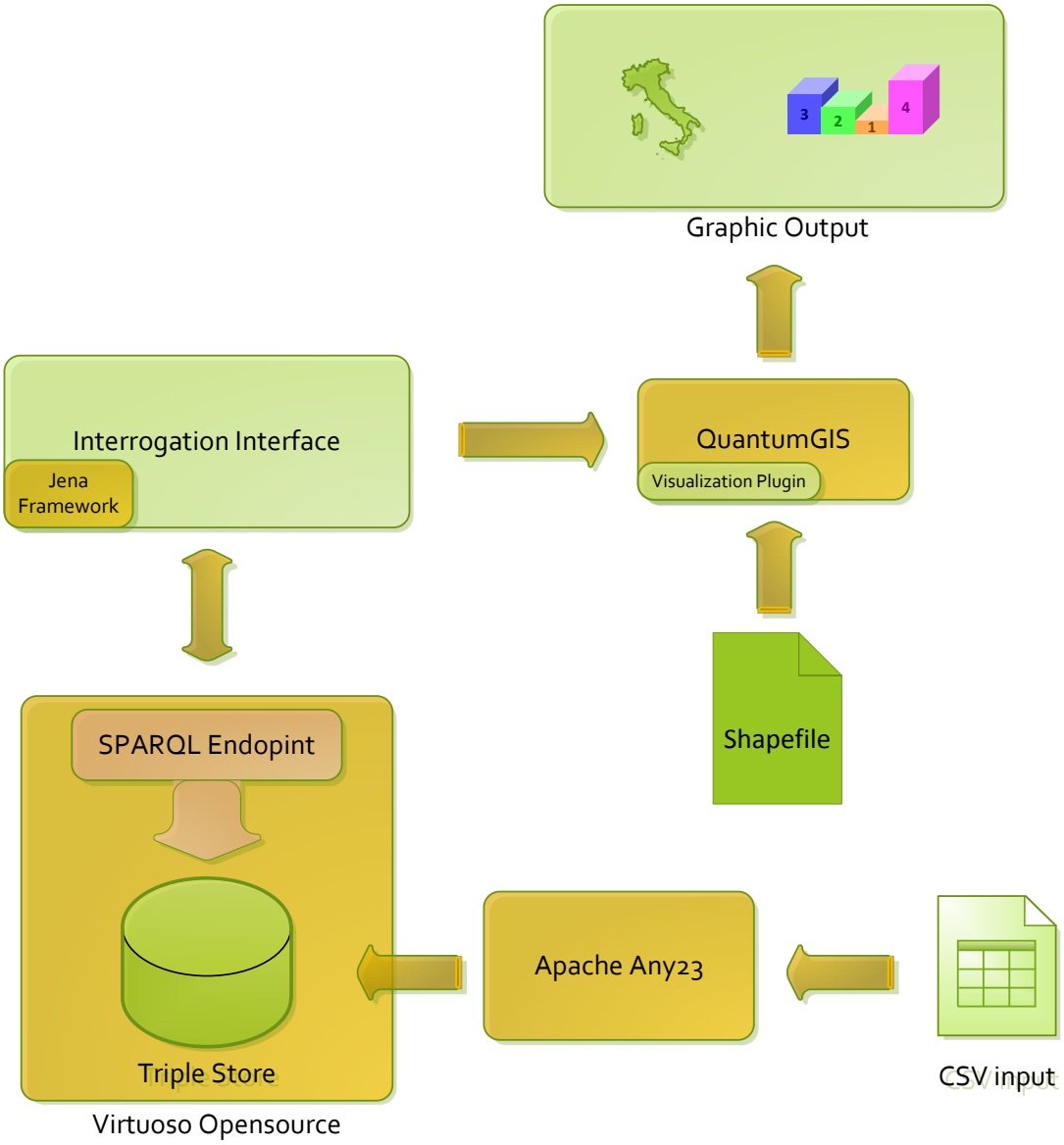

Figura 4.2: visione architetturale della soluzione

In **Figura 4.2** è mostrata una rappresentazione architetturale del sistema realizzato. L'output grafico, che consiste in una serie di grafici su elaborazioni statistiche o in punti geolocalizzati e mostrati su mappa, è creato a partire da i risultati prodotti dall'interrogazione, effettuata tramite SPARQL, sul triple store in cui sono stati memorizzati i dataset. I risultati dell'interrogazione sono elaborati insieme alle mappe vettoriali fornite dalle amministrazioni comunali. In genere le mappe vettoriali fornite sono di varia natura e mostrano informazioni diverse. Il motivo è di costruire una propria visualizzazione della mappa combinando in modo diverso le singole mappe. Infatti il sistema è pensato per visualizzare ognuna di queste mappe come un singolo layer in maniera del tutto simile ad un programma di fotoritocco. In questo modo si possono unire informazioni da diverse sorgenti per ottenere la visualizzazione desiderata. Per facilitare tale lavoro è stata sviluppata, come già accennato, una estensione per QuantumGIS che si appoggia sulle interfacce per la creazione di plugin.

I dati usati per la visualizzazione sono il prodotto dell'elaborazione di query SPARQL richieste al triple store. Tale interrogazione è effettuata tramite una applicazione realizzata per l'occasione che si basa su Jena Framework. Tale applicazione invia richieste di interrogazione presso il triple store gestito da Virtuoso Opensource e permette di esportare i risultati per un uso successivo. Nel nostro caso, l'uso riguarda proprio la resa grafica all'interno di QuantumGIS.

Affinché siano possibili le elaborazioni su triple store, è necessaria una fase di inserimento dei dati. I dati devono essere in formato RDF e sono stati prodotti principalmente a partire dalle informazioni messe a disposizione dalle amministrazioni comunali prese in esame, e pubblicate sui rispettivi portali, ma anche da servizi e fonti esterne. La trasformazione è stata effettuata tramite l'uso di Apache Any23.

4.2 Realizzazione

I dati prodotti dalle diverse amministrazioni, allo stato attuale, oltre ad una completa discrezionalità sulla scelta dei dati da pubblicare tra quelli a loro disposizione, non hanno vincoli sul modo in cui li stessi devono essere pubblicati: nessun vincolo di

formato, di struttura o di granularità. Il risultato è una sostanziale anarchia, anche all'interno della stessa amministrazione e, quindi, molto spesso grossi limiti nel cercare di unire informazioni provenienti da ambiti diversi nel tentativo di confrontare dati simili relativi ad aree diverse, o cercare correlazioni tra dati relativi ad eventi diversi insistenti sulla stessa area geografica.
Di seguito è presentata la descrizione della realizzazione del progetto, oggetto di questo documento, attraverso i passaggi necessari per far si che i dati disponibili possano essere consumati facilmente.

4.2.1 Trasformazione dei dati in formato RDF

La prima fase del progetto è consistita in una analisi dei dati disponibili. Per dati disponibili non ci si riferisce soltanto a quelli presenti sul portale del comune. Ma, a partire da quelli, si è cercato di ipotizzare dei casi d'uso, che sarebbero potuti essere di interesse per diverse figure di persone o enti, anche per lo stesso ente fornitore dei dati.
Partire dall'ipotizzare tali casi d'uso si è reso necessario per scegliere quali dataset creare per primi. In realtà ipotizzare un utilizzo dei dati è una operazione conseguente alla trasformazione ed accessibilità dei dati, operazione che dovrebbe essere eseguita comunque, proprio per far si che gli utenti siano anche ideatori e creatori di ipotesi d'uso dei dati stessi.
Ad esempio, i dati sulla composizione della popolazione, per età o per provenienza (italiani o stranieri) sono alcuni dei dati più diffusi tra quelli messi a disposizione dalle amministrazioni comunali, sia perché sono dati storicamente già pubblici, anche se in passato non disponibili on-line, e quindi dati che possono essere facilmente resi disponibili senza che tali informazioni creino particolari problemi di privacy se opportunamente aggregati, sia perché riguardano la popolazione stessa, cioè l'utilizzatore finale di tali dati.
In questo scenario si inseriscono, quindi, tutte quelle query tese ad esplorare la composizione e localizzazione della popolazione in modo da poter programmare la fornitura di servizi legati alla densità abitativa, come uffici pubblici, o legati in modo specifico all'età, come servizi scolastici o di assistenza agli anziani, e a posizionarli sul territorio in modo da garantire maggiore vicinanza e copertura per i possibili

utenti in funzione, quindi, della modalità di distribuzione della categoria di persone oggetto della ricerca, o verificare se la distribuzione di tali servizi rispecchi in qualche modo la distribuzione dei possibili utilizzatori. Ad esempio conoscere in quale area è presente la maggiore densità di bambini in età prescolare e quale/i asilo/i sono presenti in quella zona, potrebbe permettere l'attivazione per tempo di nuove classi o spostamento di quelle classi presso altra struttura che ne avesse bisogno.

Per poter soddisfare tali ipotesi di scenari, in alcuni casi, è stato necessario rivolgersi, per la ricerca di dati da unire, a sorgenti di dati esterne a quelle comunali. Tale operazione è stata utile per affrontare il discorso della verifica della complessità di integrazione di dati con origini differenti.

Una volta scelti i dati si è presentato il problema di trasformare tali dati. La maggior parte di loro erano già in origine in formati tabellari, altri, come sarà mostrato nel capitolo successivo relativamente all'esplorazione di un reale caso d'uso, erano in origine parti di pagine Web che avevano come utente finale una persona e non una macchina o un software per l'estrazione di informazioni.

Per questo motivo, in alcuni casi, si è resa necessaria una resa di tali informazioni in modo che lo strumento scelto per la trasformazione in formato RDF, ovvero Any23 descritto nel capitolo precedente, potesse eseguire con successo tale trasformazione. In sostanza, essendo stato scelto questo strumento come vincolo di progetto, è stato necessario adattare i dati da elaborare alle sue specifiche, specifiche in ogni caso ragionevoli per uno strumento che non necessita di particolari configurazioni per eseguire la trasformazione.

Quindi, per editare e riorganizzare i file di dati iniziali è stato usato Google Refine [36]. Uno strumento sviluppato da Google che ha come scopo di supportare proprio tale tipo di lavoro. Ovvero di supportare la modifica di dati che, soprattutto quando si ha a che fare con dati aperti e prodotti senza particolare cura dei dettagli, debbano subire delle modifiche o aggiustamenti prima di poter essere utilizzati con profitto.

Google Refine è dotato di caratteristiche tali da permettere di rieditare e riconfigurare file di dati organizzati in maniera confusa e non necessariamente in formati tabellari. Ciò è eseguito attraverso la fornitura di semplici API per creare comandi da poter applicare in maniera automatica alle varie righe e/o colonne. A partire dal file originale, attraverso passaggi consecutivi, è possibile giungere ad un documento tale

da soddisfare i vincoli imposti per effettuare con profitto la trasformazione in formato RDF.

In questo passaggio, i problemi riscontrati più spesso riguardano, ad esempio, l'assenza di valori in alcune celle della tabella, di norma ad indicare che il valore in tal caso è uguale a zero. Purtroppo, in tal caso Any23, interpreta tale configurazione non come assenza di valore ma come inserimento errato di un ulteriore separatore, facendo così corrispondere erroneamente proprietà e rispettivi valori.

Altro caso è la presenza di più righe di intestazione all'interno del file CSV. Come detto già nel capito precedente, presentando lo strumento Any23, la prima riga di etichette delle colonne, viene usata per etichettare le proprietà trasformate. Ciò ha comportato o l'eliminazione delle righe in eccesso o, nel caso fosse stato necessario, l'unione delle informazioni in un'unica riga.

```
FARMACIA AI COLLI - Piazza Di Porta Castiglione, 15/E - 40136 - BOLOGNA
FARMACIA AICARDI - Via S.Vitale, 58 - 40125 - BOLOGNA (No CUP)
FARMACIA AL PALAZZO DELLO SPORT - Via Delle Lame, 52 - 40122 - BOLOGNA (No CUP)
FARMACIA AL SACRO CUORE - Via Matteotti, 29 - 40129 - BOLOGNA
FARMACIA AL VELODROMO - Via Vittorio Veneto, 19 - 40131 - BOLOGNA
FARMACIA ALBERANI - Via Farini, 19 - 40124 - BOLOGNA
FARMACIA ANTICA DEI SERVI - Strada Maggiore, 39 - 40125 - BOLOGNA (No CUP)
FARMACIA ANTICA DELLE MOLINE - Via A. Righi, 6/A - 40126 - BOLOGNA
FARMACIA BARTOLOTTI - Via Fioravanti, 26 - 40129 - BOLOGNA
FARMACIA BEATA VERGINE DI S. LUCA - Via D'Azeglio, 15 - 40123 - BOLOGNA
FARMACIA BERTELLI ALLA FUNIVIA - Via Porrettana, 95 - 40135 - BOLOGNA
FARMACIA BETTINI - Via Di Corticella, 68 - 40128 - BOLOGNA
FARMACIA BUSACCHI - Via Emilia Ponente, 24 - 40133 - BOLOGNA (No CUP)
FARMACIA CARRACCI - Piazza Liber Paradisus, 14/15 - 40129 - BOLOGNA (No CUP)
FARMACIA CASTIGLIONE - Via Castiglione, 53 - 40124 - BOLOGNA (No CUP)
FARMACIA CHILLEMI - Via Bellaria, 36 - 40139 - BOLOGNA
FARMACIA COMUNALE ANDREA COSTA - Via Andrea Costa, 156/1-2 - 40134 - BOLOGNA
FARMACIA COMUNALE ARNO - Via Arno, 36/A - 40139 - BOLOGNA
FARMACIA COMUNALE AZZURRA - Via Azzurra, 52/2 - 40138 - BOLOGNA
```

Figura 4.3: parte della lista delle farmacie di Bologna

Il caso più articolato riguarda, naturalmente, la creazione di un file tabellare a partire da righe di testo senza particolari separatori. Google Refine offre funzioni in grado di definire espressioni regolari sul testo e di individuare separatori in base ad essi. Tale problema si è verificato quando è stato prelevato l'elenco di attività commerciali e indirizzi di tali attività direttamente da una pagina Web. Comprensibilmente non era

possibile utilizzare direttamente tali informazioni ma necessitavano proprio di questo lavoro di modifica e riconfigurazione.

Ad esempio prendiamo il contenuto della **Figura 4.3**. Qui è mostrata una parte della lista delle farmacie di Bologna. Tale estratto è preso direttamente da [37] dove è presente la lista delle farmacie di tutta la provincia. Come si può vedere, si tratta di una semplice lista testuale, non è presente alcuna divisione in colonne, ma solo una normale lista.

Per far si che tale lista possa essere trasformata con profitto da Any23 è necessario definire una tabella e quindi dividere le righe in modo tale che si possano definire delle proprietà a partire dal nome delle colonne. Per effettuare questa trasformazione, Google Refine mette a disposizione degli strumenti pensati per questo genere di operazioni:

- è possibile dividere una colonna, in questo caso in origine la colonna era unica, in più colonne sulla base di un separatore testuale, definendo anche un numero di colonne in cui deve essere diviso. Oppure in base al numero dei caratteri. In questo caso tale operazione è stata ripetuta per effettuare una divisione usando come separatore il segno "-" e poi la "," per separare il nome della via dal numero civico;
- effettuare una sostituzione dei valori di una colonna attraverso l'uso di un linguaggio di script GREL [38] che permette di applicare trasformazioni e sostituzioni al contenuto delle celle in base all'aderenza del contenuto stesso ad un pattern definito dall'utente. In questo modo è risultato semplice far si che a tutti i record della tabella fosse inserita una informazione sul servizio CUP;
- permette di trasporre parte della tabella in modo sia da trasformare colonne in nuove righe o, viceversa, righe in nuove colonne. Questa trasformazione permette di riconfigurare facilmente la logica seguita per modellare i dati;
- attraverso l'uso di viste specifiche sulla tabella, denominate *facets*, ad esempio basate sui valori di una colonna selezionata, è possibile applicare le operazioni descritte solo su una parte delle righe della tabella offrendo, quindi, una maggiore versatilità.

In **Figura 4.4** è mostrato il risultato dell'elaborazione prodotta a partire dai dati del precedente esempio di **Figura 4.3**. L'operazione è stata portata a termine dividendo le righe in colonne facendo uso delle operazioni appena descritte.

Nome Farmacia	Indirizzo	Numero	CAP	Città	CUP
FARMACIA AI COLLI	Piazza Di Porta Castiglione	15	40136	BOLOGNA	true
FARMACIA AICARDI	Via S.Vitale	58	40125	BOLOGNA	false
FARMACIA AL PALAZZO DELLO SPORT	Via Delle Lame	52	40122	BOLOGNA	false
FARMACIA AL SACRO CUORE	Via Matteotti	29	40129	BOLOGNA	true
FARMACIA AL VELODROMO	Via Vittorio Veneto	19	40131	BOLOGNA	true
FARMACIA ALBERANI	Via Farini	19	40124	BOLOGNA	true
FARMACIA ANTICA DEI SERVI	Strada Maggiore	39	40125	BOLOGNA	false
FARMACIA ANTICA DELLE MOLINE	Via Augusto Righi	6	40126	BOLOGNA	true
FARMACIA BARTOLOTTI	Via Fioravanti	26	40129	BOLOGNA	true
FARMACIA BEATA VERGINE DI S. LUCA	Via D'Azeglio	15	40123	BOLOGNA	true
FARMACIA BERTELLI ALLA FUNIVIA	Via Porrettana	95	40135	BOLOGNA	true
FARMACIA BETTINI	Via Di Corticella	68	40128	BOLOGNA	true
FARMACIA BUSACCHI	Via Emilia Ponente	24	40133	BOLOGNA	false
FARMACIA CARRACCI	Piazza Liber Paradisus	14	40129	BOLOGNA	false
FARMACIA CASTIGLIONE	Via Castiglione	53	40124	BOLOGNA	false
FARMACIA CHILLEMI	Via Bellaria	36	40139	BOLOGNA	true
FARMACIA COMUNALE ANDREA COSTA	Via Andrea Costa	156	40134	BOLOGNA	true
FARMACIA COMUNALE ARNO	Via Arno	36	40139	BOLOGNA	true
FARMACIA COMUNALE AZZURRA	Via Azzurra	52	40138	BOLOGNA	true

Figura 4.4: risultato della trasformazione con Google Refine dell'esempio di **Figura 4.3**

```
<http://www.salute.bologna.it/Elenco_farmacie_Bologna#row/0>
        a <http://vocab.sindice.net/csv/Row> ;
        <http://www.salute.bologna.it/Elenco_farmacie_Bologna#NomeFarmacia>
                "FARMACIA AI COLLI"^^<http://www.w3.org/2001/XMLSchema#string> ;
        <http://www.salute.bologna.it/Elenco_farmacie_Bologna#Indirizzo>
                "Piazza Di Porta Castiglione"^^<http://www.w3.org/2001/XMLSchema#string> ;
        <http://www.salute.bologna.it/Elenco_farmacie_Bologna#Numero>
                "15"^^<http://www.w3.org/2001/XMLSchema#integer> ;
        <http://www.salute.bologna.it/Elenco_farmacie_Bologna#Cap>
                "40136"^^<http://www.w3.org/2001/XMLSchema#integer> ;
        <http://www.salute.bologna.it/Elenco_farmacie_Bologna#CittÃfâ >
                "BOLOGNA"^^<http://www.w3.org/2001/XMLSchema#string> ;
        <http://www.salute.bologna.it/Elenco_farmacie_Bologna#Cup>
                "true"^^<http://www.w3.org/2001/XMLSchema#string> .
```

Figura 4.5: risultato dell'elaborazione da parte di Apache Any23 della prima riga dell'esempio di **Figura 4.4**

Completata questa fase è stato possibile produrre con facilità dei file RDF in codifica turtle che potessero essere oggetto dell'elaborazione da parte del servizio web in grado di supportare interrogazioni sugli stessi.

Durante la creazione Any23 richiede di indicare un URI che utilizza per definire un namespace all'interno del dataset creato. Ad ogni record e proprietà è associato tale namespace a meno che l'etichetta della colonna non risulti già essere un URI. A scopo indicativo gli URI scelti sono stati costruiti a partire dall'indirizzo del rispettivo portale della pubblica amministrazione da cui sono stati prelevati. Ad esempio, in **Figura 4.5** è stato usato come URI per definire il namespace relativo alle farmacie di Bologna: "http://www.salute.bologna.it/Elenco_farmacie_Bologna#".

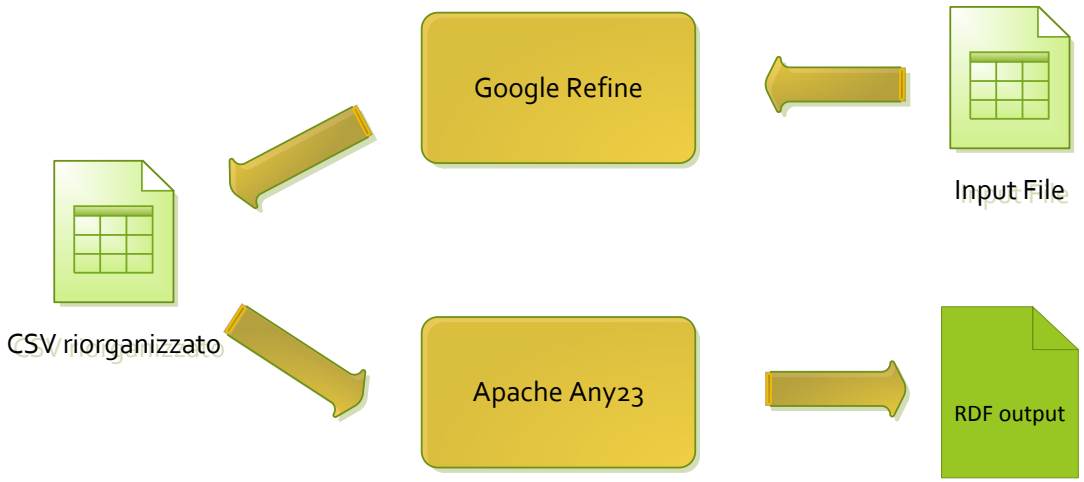

Figura 4.6: schema di funzionamento della serializzazione dei dati in RDF

In **Figura 4.6** è mostrato lo schema di funzionamento descritto fin qui. Quindi, volendo riassumere il discorso, quello che in figura è indicato come Input file, non deve essere necessariamente un file con dati in formato tabellare. Le capacità offerte da Refine permettono di ottenere con buona semplicità un file adatto ad essere trasformato RDF. Any23 permette anche di scegliere il formato di serializzazione RDF in output. La scelta di default è stata Turtle perché, come detto in precedenza, risulta essere il formato meno verboso ed anche più veloce da processare dai diversi parser.

Come detto nel capitolo precedente, a riguardo dello strumento Any23, sarebbe stato possibile sostituire alcune intestazioni di colonne, in particolare quelle relative a concetti comuni, con URI definiti a priori. Tale operazione esula dall'attività qui

svolta perché si sarebbe dovuta svolgere una operazione di individuazione di tali concetti comuni a priori che avrebbe richiesto un analisi più approfondita sia dei vari dataset che del modo di pubblicazione degli stessi tra i vari enti proprietari dei dati. Invece si è scelto di mantenere la struttura dei dati il più simile possibile all'originale anche per valutare i limiti di tale approccio.

4.2.2 Interfaccia remota di interrogazione

I file RDF in formato turtle prodotti sono stati inseriti all'interno della funzionalità di memorizzazione di grafi RDF messo a disposizione da Virtuoso Opensource. In questo modo è stato possibile far si che i dataset trasformati potessero essere interrogati ed elaborati attraverso query SPARQL.

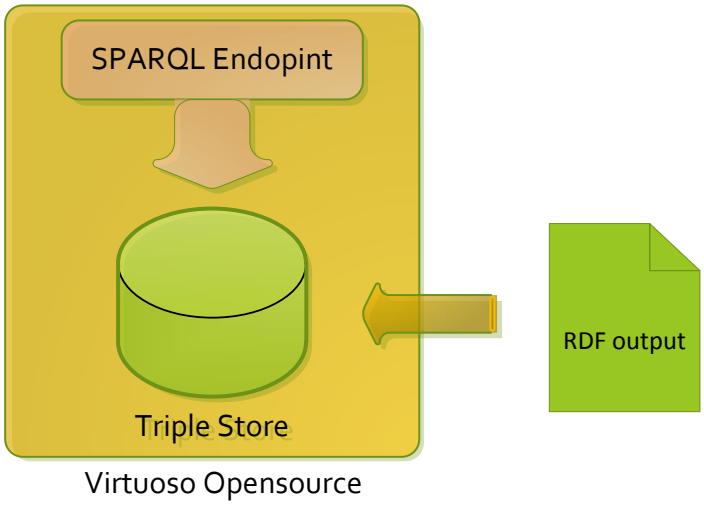

Figura 4.7: popolamento del triple store

L'interfaccia di gestione di Virtuoso, ovvero Virtuoso Conductor, una interfaccia web che permette l'amministrazione remota di tutte le funzionalità del servizio, consente di effettuare l'upload dei dataset all'interno del servizio (**Figura 4.7**) permettendo di specificare un nome per ogni grafo caricato. Inoltre è possibile dare a due grafi uno stesso nome se si intende effettuare, con una chiamata ad un unico grafo, una interrogazione su più di un dataset.

Virtuoso Opensource integra, inoltre, una interfaccia di interrogazione remota attraverso protocollo HTTP. Tale interfaccia è interrogabile anche attraverso una apposita pagina Web messa a disposizione dal servizio stesso. Si è optato per una interrogazione tramite un'applicazione dedicata perché avrebbe permesso più

flessibilità e customizzazione, sia per di vista dell'input che dell'output, che non sarebbe stato possibile attraverso l'utilizzo dell'interfaccia Web.

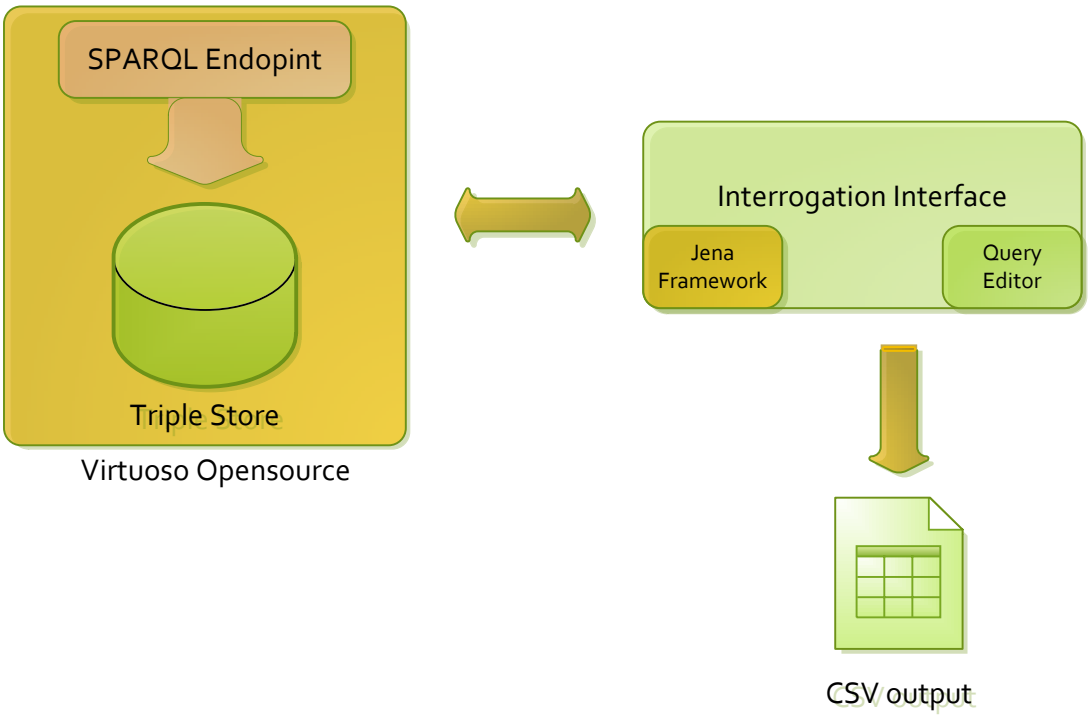

Figura 4.8: interrogazione del triple store tramite SPARQL

L'applicazione creata è stata sviluppata in Java e si basa su Jena Framework per l'interrogazione di endpoint SPARQL (**Figura 4.8**). Tale applicazione è stata pensata, prima di tutto, per fornire accesso a diversi endpoint SPARQL disponibili. È, infatti, possibile configurare tale applicazione per interrogare qualsiasi endpoint SPARQL disponibile sul Web. In questo modo è stato possibile accedere e recuperare informazioni già disponibili in rete in formati RDF ed utilizzate per integrarle con i dati forniti delle pubbliche amministrazioni iniziali.

L'interfaccia realizzata è costituita da una parte di inserimento dei dati, in cui da tastiera è possibile inserire il testo costituente la query, e da una finestra di output dei risultati. Dispone anche di una finestra in cui sono indirizzati i messaggi di errore provenienti dal servizio di endpoint SPARQL per permettere un debug delle query.

L'interfaccia dispone, inoltre, della capacità di salvare e caricare su file le query prodotte, ed è in grado di interrogare anche altri servizi di endpoint SPARQL oltre a quello messo a disposizione in questo progetto. Sono già preimpostati e selezionabili i servizi messi a disposizione da DBpedia [39] e Linked Open Data Italia: Scuole

Italiane [40]. Inoltre è disponibile un campo di testo per selezionare un indirizzo per un servizio diverso.

Il sistema prevede, come descritto nel capitolo precedente circa il funzionamento del framework Jena, a partire della pressione del tasto "Run!", l'invocazione del metodo che effettua la costruzione di un oggetto "`Query`" a partire dalla stringa di testo passata attraverso l'area di inserimento (ovvero la query vera e propria). In seguito i risultati vengono prelevati dall'oggetto risposta "`ResultSet`" dopo l'invocazione della query sul servizio selezionato attraverso la classe factory "`QueryExecutionFactory`". Dall'oggetto di risposta, i risultati vengono poi formattati per essere visualizzati o salvati su file con alcune possibilità diverse tra cui il formato CSV. Oltre alle caratteristiche fin qui descritte, è possibile anche salvare o caricare da file il testo di una query. Allo stato attuale il sistema supporta solo query di tipo SELECT.

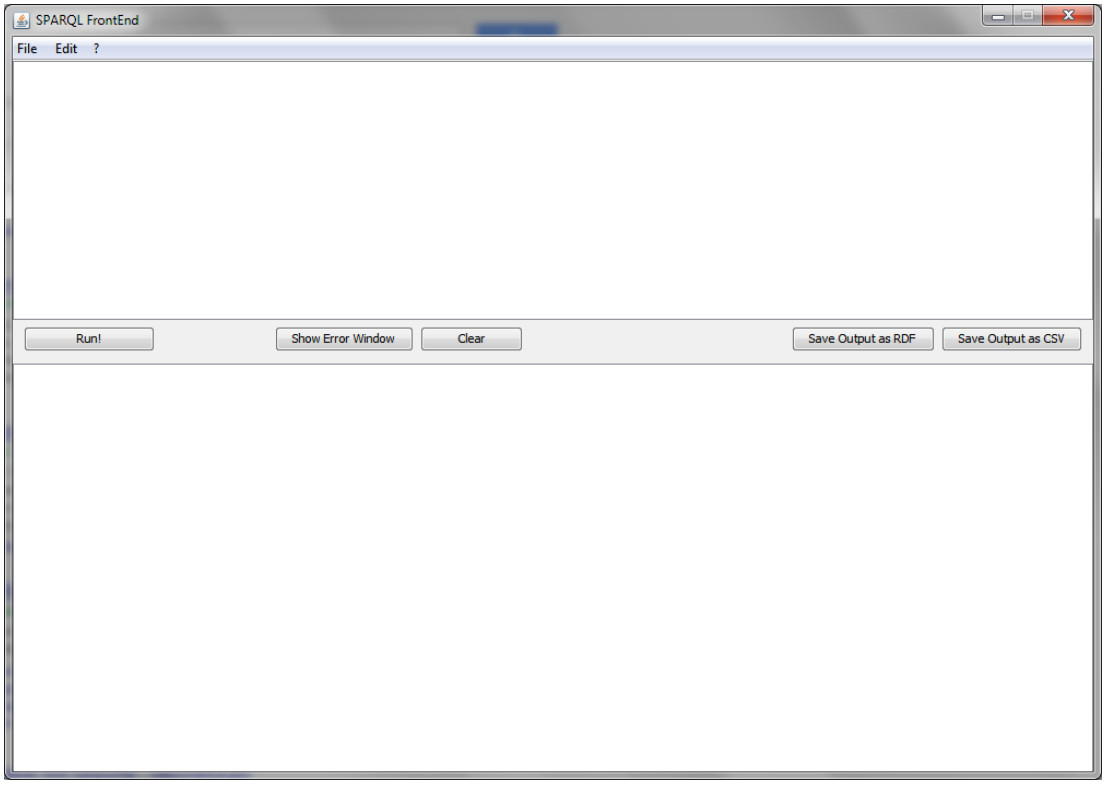

Figura 4.9: interfaccia dell'applicazione realizzata

In **Figura 4.9** è mostrata l'interfaccia dell'applicazione realizzata. Sono presenti due campi di testo principali. In quello superiore è possibile inserire la query che si vuole richiedere. In quello inferiore viene presentato in output il risultato della query. Il

risultato viene visualizzato sotto forma di tabella grazie alla classe fornita da Jena Framework "`ResultSetFormatter`". Tale classe dispone di soli metodi statici che permettono di trasformare il risultato della query in diversi formati di output tra cui semplice testo ("`ResultSetFormatter.asText()`"), CSV oppure in diversi formati di serializzazione RDF. Questa stessa classe è stata usata per permettere di salvare il risultato su file come CSV e RDF (rispettivamente con i metodi `ResultSetFormatter.outputAsCSV()` e `ResultSetFormatter.outputAsRDF()`) questi metodi richiedono in *signature* uno stream di output per completare il salvataggio. A questi metodi viene passato al momento della chiamata l'oggetto "`ResultSet`" che contiene il risultato della query.

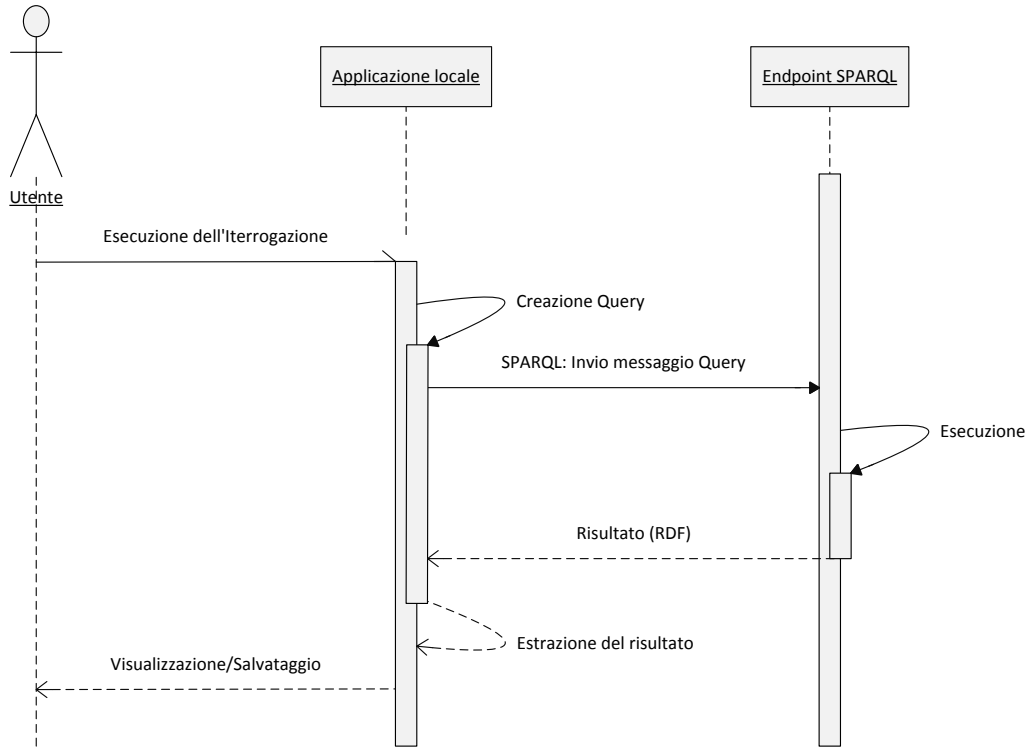

Figura 4.10: diagramma di sequenza di una interrogazione all'endpoint SPARQL

La classe "`ResultSet`", però, ha la caratteristica di cancellare il contenuto una volta richiamato il risultato; scelta fatta a livello progettuale dagli sviluppatori di Jena per non occupare spazi di memoria potenzialmente molto ampi. Al momento dell'arrivo del risultato della query questo viene subito visualizzato. Per permettere di salvare il risultato su file, per evitare di dover eseguire di nuovo la query, è stato necessario

utilizzare la classe "`ResultSetRewindable`". Tale classe non cancella il risultato e può essere recuperato in seguito, ed è, quindi, adatto ai nostri scopi.

In **Figura 4.10** è mostrato un diagramma di sequenza in cui è descritta l'attività di interrogazione presso il triple store remoto.

A questo punto, si è proseguito ipotizzando e realizzando delle query SPARQL per completare i casi d'uso ipotizzati in precedenza, di modo da riuscire a combinare insieme informazioni per ottenere nuove informazioni.

È stato preso in esame uno specifico caso su richiesta del committente, che per l'occasione ha provveduto a fornire anche dei dataset più dettagliati, ovvero il caso delle Politiche 2013 e del confronto, per zona, tra i risultati elettorali nelle sezioni del comune di Bologna e alcune delle informazioni disponibili sulla popolazione della stessa città. Per questo caso specifico sono state create dei modelli di query che vengono completati tramite una interfaccia grafica tramite menù a tendina.

La creazione di tali modelli di query è partita dall'individuazione di alcuni modelli comuni di per le specifiche informazioni seguita dall'inserimento di parametri all'interno di tali query in modo che, in base alla selezione dell'utente, permettesse di costruire una query corretta sia sintatticamente che semanticamente. In questo modo l'esecuzione della query risultante produce il risultato conforme alle richieste iniziali.

In relazione ad i dataset su popolazione e risultati elettorali sono stati approntati diversi scheletri di query:

- interrogazione in merito al risultato ottenuto dal singolo partito: la query è costruita in modo da selezionare, in base alla scelta dell'utente, il partito (fino a tre partiti in un'unica query) di cui si vuole conoscere la performance, inoltre è possibile selezionare la camera e il livello di aggregazione del risultato. I dati originali hanno come livello di aggregazione il seggio elettorale, per cui è stato possibile ottenere risultati anche livelli di aggregazione maggiori;
- interrogazione per individuare l'astensionismo: anche in questo caso è possibile individuare l'andamento dell'astensionismo per le diverse zone e con livelli di aggregazione differente oltre che per le due camere. Anche qui i risultati pubblicati sono relativi ad i singoli seggi elettorali;

- informazioni sui redditi della popolazione: a partire dai dati sul reddito forniti esclusivamente, trai i comuni in esame, dal comune di Bologna è stato possibile risalire al dato di reddito medio per contribuente per le diverse zone della città. Il livello di aggregazione con cui sono stati rilasciati questi dati è di granularità più grossa rispetto ad i risultati elettorali ma più piccola rispetto agli atri dataset sulla popolazione. Inoltre non è presente nessuna informazione sulla relazione che intercorre tra le aree su cui insistono questi dati e quelli elettorali, ovvero le sezioni elettorali, per cui per poter fare un confronto è stato necessario utilizzare come livello minimo di granularità un livello di aggregazione maggiore di cui si avesse a disposizione un informazione di contenimento per entrambi i dataset;
- informazioni sulle fasce di età della popolazione: a partire dai dataset contenenti informazioni sul numero di persone residenti in specifiche zone della città e della loro divisione in fasce di età è stato possibile ottenere questa informazione per diversi livelli di aggregazione e per diverse fasce di età;
- le interrogazioni fin qui presentate sono state pensate per poter essere eseguite sia singolarmente, ma anche combinate in modo da poter avere in un'unica tabella i risultati dei partiti con una delle altre query realizzate, naturalmente con lo stesso livello di aggregazione. Lo scopo è stato quello di individuare eventuali preferenze nel voto per i diversi gruppi di popolazione. Ciò è stato ottenuto costruendo una nuova query che utilizzasse le altre già fatte come sotto-query. Tali query, essendo costituite da più di un livello di sotto-query, sono risultate essere anche quelle che hanno avuto un tempo di esecuzione maggiore delle altre.

La creazione della query è demandata alla classe "`QueryEditor`". Un componente dell'applicazione che, attraverso il metodo "`creaQuery()`", costruisce la query a partire dalle scelte dell'utente.

La query prodotta è visualizzata all'interno del campo di input dell'interfaccia. In questo modo un utente un po' più esperto può esaminare la query prodotta ed eventualmente modificarla e personalizzarla secondo le proprie esigenze in maniera da ottenere risultati diversi da quelli delle query preimpostate.

In **Figura 4.11** è mostrata l'interfaccia di selezione che permette di selezionare e generare una query relativa alle elezioni politiche a partire dai dati presenti. A seconda della selezione della città e della camera, i campi relativi ai partiti presenti in lista vengono popolati automaticamente. Ciò avviene tramite l'interrogazione del sistema in maniera trasparente all'utente. Ciò comporta, però, che il servizio deve essere attivo e raggiungibile affinché la query possa essere generata. In caso contrario non è possibile ottenere la lista dei partiti che dovrà essere utilizzata per la creazione.

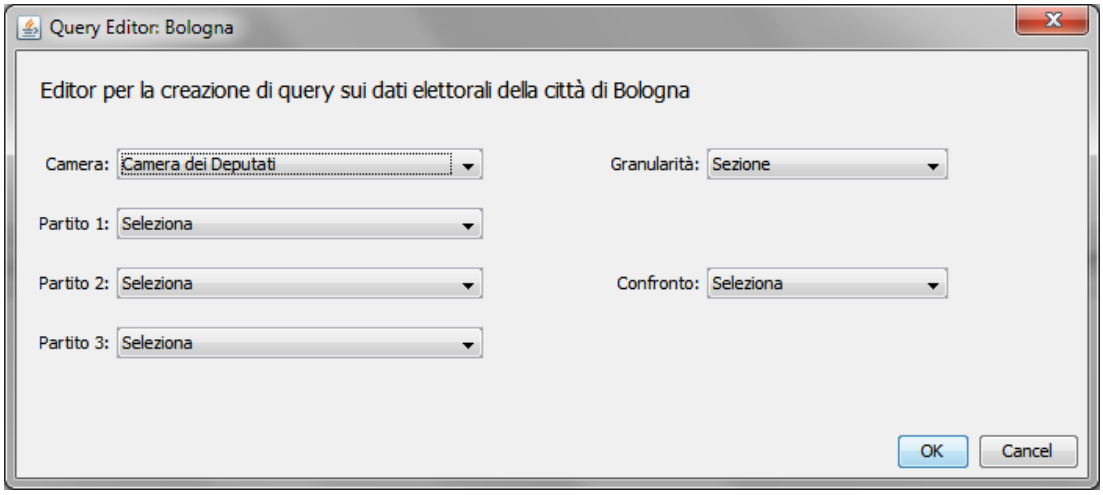

Figura 4.11: interfaccia di selezione per la creazione di query per le elezioni Politiche

A questo punto un utente non esperto può facilmente ottenere informazioni che permettono di confrontare l'andamento di alcuni partiti sulle varie zone del territorio comunale assieme ad informazioni sulla popolazione. Tale operazione è stata effettuata anche per i risultati elettorali delle Politiche 2013 per i comuni di Firenze e Roma. La descrizione più dettagliata del caso d'uso è proposta nel capitolo seguente.
Come già detto, il risultato dell'interrogazione può essere salvato come CSV. In tal modo, i dati prodotti così strutturati possono essere visualizzati graficamente. Dal momento che molte di queste informazioni sono legate al territorio, è stato usato QuantumGIS per produrre visualizzazioni grafiche dei risultati.

4.2.3 Visualizzazione su mappa

Tra i dati delle amministrazioni comunali sono spesso presenti anche dati cartografici, in particolare comune è la presenza di file, in formato Shapefile, riportanti i confini del comune e la divisione in quartieri. In questo modo è possibile

mostrare grafici su dati statistici, elaborati tramite le query precedentemente eseguite, localizzati sulle rispettive aree a cui i dati originali fanno rifermento.

Lo strumento QuantumGIS fornisce le funzionalità in grado di permettere una visualizzazione di questo tipo. Per far ciò è sufficiente aggiungere i dati prodotti nella precedente fase ed associarli ciascuno alla zona di riferimento, ovvero alla figura geometrica (*feature* secondo la terminologia GIS).

Se, ad esempio, i risultati di un analisi sui dataset sono stati aggregati per quartiere, di tali dati è possibile effettuare un incrocio di tali dati con i dati tabellari associati a ciascuna feature rappresentante un quartiere dello shapefile. A questo punto è possibile rappresentare grafici a partire da tali dati sul layer interessato. Questo permette di avere un colpo d'occhio migliore sui dati e permette di individuare più facilmente eventuali pattern sui dati analizzati.

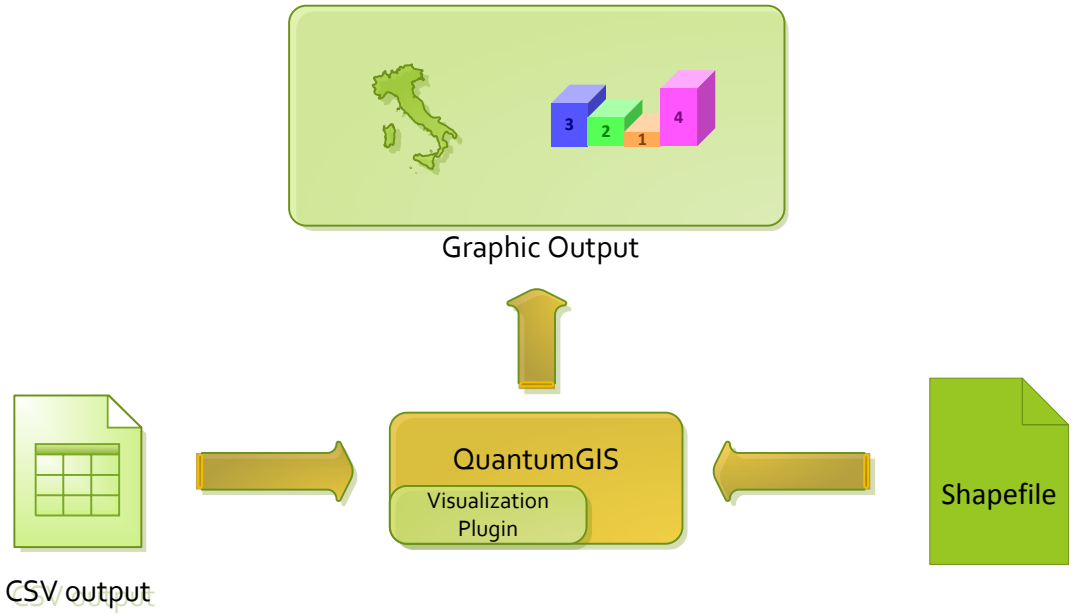

Figura 4.12: visualizzazione su mappa tramite il plugin per QuantumGIS

In questo caso, per fornire un supporto all'utente inesperto, è stata costruita una estensione per QuantumGIS (**Figura 4.12**). Come già detto, tale strumento supporta estensioni scritte in Python. A partire dalle API messe a disposizione, è possibile creare dei plug-in grado di supportare l'utente nelle operazioni di interesse.

Le API [41] permettono di accedere all'ambiente di lavoro dell'applicazione, e di interagire con lo stesso. Sia caricando o eliminando layer dall'ambiente di lavoro, sia

modificando gli stessi, creandone di nuovi o eseguendo operazioni di analisi ed elaborazioni dei dati dei layer.

L'estensione realizzata ha avuto come obiettivo quello di riunire in un'unica posizione e semplificare l'operazione di unione dei dati prodotti dalle query SPARQL, del passo di elaborazione precedente, con le feature dei layer e la costruzione di grafici.

L'operazione di unione (*join* tra le due tabelle) dei dati è basata sull'individuazione di valori identici tra le celle delle colonne selezionate dall'utente, delle due rispettive tabelle dei file di input. Tale operazione presuppone che ciascuno dei due file, shapefile della mappa e CSV risultato delle query SPARQL, abbiano una colonna con degli identificatori unici.

COD_QUAR	QUAR_NOME	COD_ZONA	ZONA_NOME
1	Borgo Panigale	C	Borgo Panigale
2	Navile	B	Bolognina
2	Navile	E	Corticella
2	Navile	I	Lame
3	Porto	M	Marconi
3	Porto	P	Saffi
4	Reno	A	Barca
4	Reno	S	Santa Viola
5	San Donato	Q	San Donato
6	Santo Stefano	D	Colli
6	Santo Stefano	G	Galvani
6	Santo Stefano	O	Murri
7	San Vitale	H	Irnerio
7	San Vitale	T	San Vitale
8	Saragozza	F	Costa Saragozza
8	Saragozza	L	Malpighi
9	Savena	N	Mazzini
9	Savena	R	San Ruffillo

Figura 4.13: codici per le zone di Bologna

Per far fronte a questo problema sono stati usati come identificatori i codici associati alle diverse zone dalle rispettive amministrazioni. In **Figura 4.13** è mostrato un esempio dei codici delle zone definiti per la città di Bologna. Inserendo tale informazione nei risultati delle query, e dal momento che gli shapefile erano già forniti di tali dati è stato possibile effettuare il join delle tabelle.

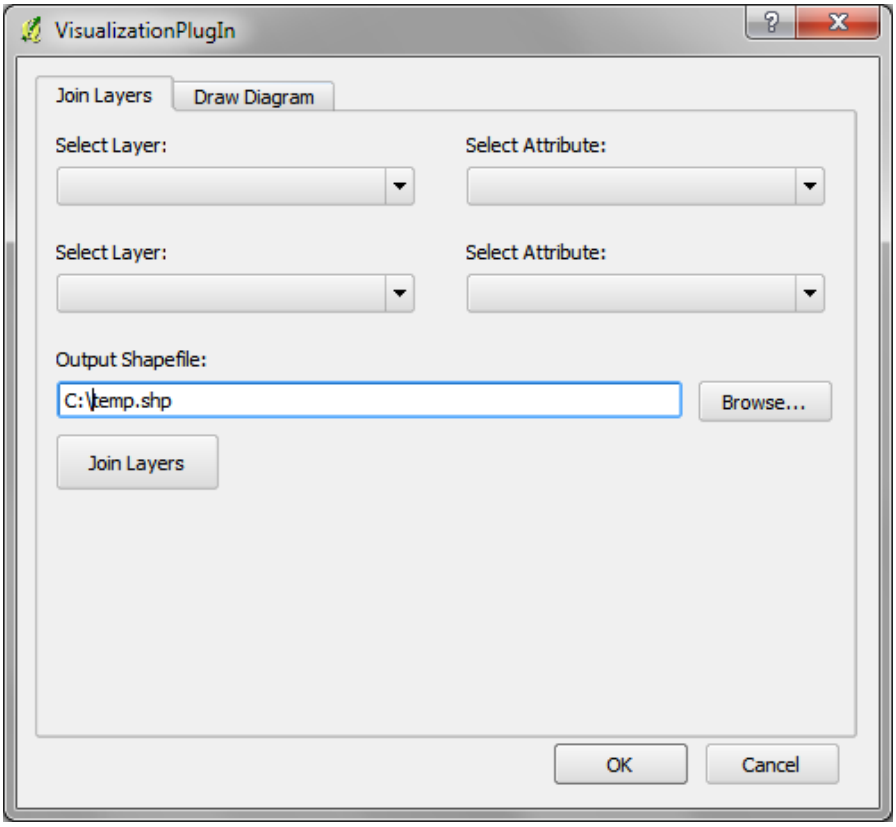

Figura 4.14: interfaccia del plugin per QuantumGIS realizzato (operazione di join dei layer)

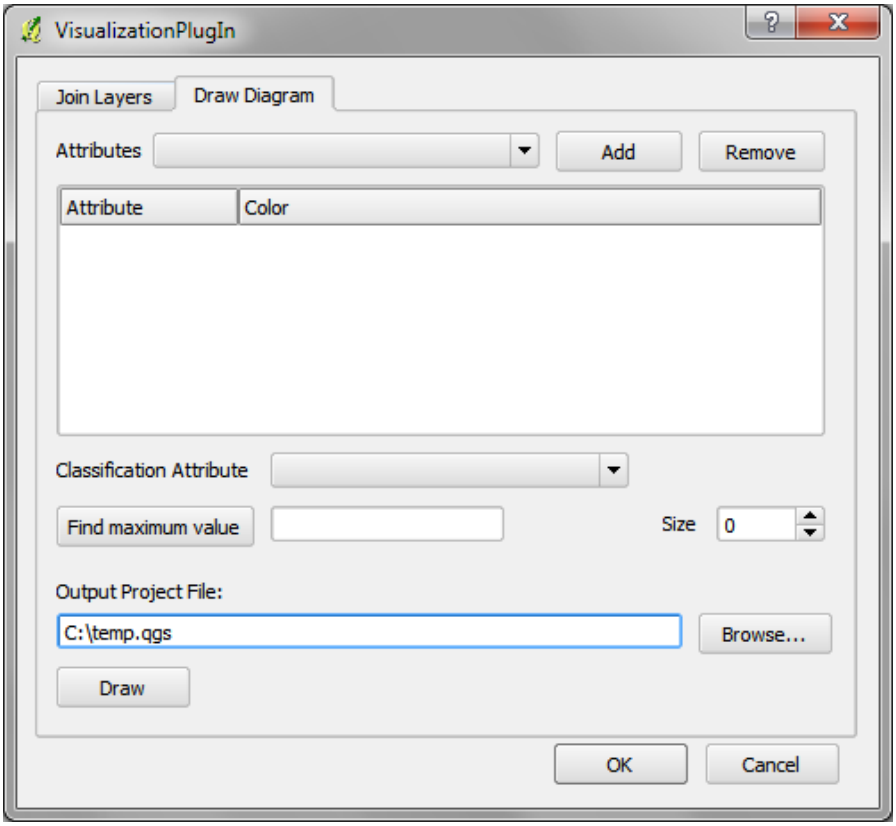

Figura 4.15: interfaccia del plugin per QuantumGIS realizzato (operazione di creazione dei grafici)

L'interfaccia creata a supporto permette di selezionare, tra i layer caricati all'interno di QuantumGIS, il layer della cartina desiderato e il CSV dei dati da unire, anch'esso caricato come layer. Per ciascuno dei due si deve selezionare la colonna su cui fare il join che consentirà di creare un nuovo shapefile con le feature aventi le informazioni aggiuntive. L'operazione di join procede secondo il seguente algoritmo:

- per ciascuna feature dello shapefile, il valore della corrispondente cella della colonna scelta per il join viene confrontato con la prima delle celle della colonna scelta per l'atra tabella;
- nel caso il confronto desse esito negativo viene selezionata la riga successiva;
- altrimenti, ovvero nel caso in cui i valori siano corrispondenti, viene creato un nuovo oggetto feature a cui è associata la geometria e i dati associati ad essa dello shapefile e sono aggiunte le colonne del file CSV con i valori del record con cui ha ottenuto un confronto dall'esito positivo;
- l'insieme delle nuove feature è usato per creare un nuovo shapefile che costituirà il join dei due file di origine;
- tale file è caricato nell'ambiente di lavoro in modo da poter essere immediatamente utilizzato.

Attraverso il metodo "`QgsMapLayerRegistry.instance().mapLayers()`" è possibile recuperare il riferimento a tutti i layer attualmente caricati nell'area di lavoro di QuantumGIS ("`QgsMapLayerRegistry.instance()`" fornisce accesso a tutte le operazioni possibili sull'area di lavoro). A partire da questo riferimento vengono create le liste che andranno a popolare i menù a tendina "Select Layer" e "Select Attribute" presenti in **Figura 4.14**. Il primo campo "Select Layer" ha il vincolo di essere costituito solo da layer vettoriali e le geometrie di tale layer verranno usate per creare le geometrie del nuovo layer creato alla fine dell'operazione di join. Effettuata la selezione del file di output, l'operazione di join, che è stata definita all'interno del metodo "`attribute_join()`", può essere eseguita. "`QgsFeature()`" permette di creare delle geometrie vuote. Queste geometrie vengono inizializzate con le geometrie del layer selezionato nel primo campo "Select Layer". Attraverso il valore selezionato in "Select Attribute" si ottiene il riferimento all'attributo di ciascun layer su cui effettuare il confronto per il join. Attraverso il

metodo "`fieldNameIndex()`" applicato ad un oggetto layer è possibile ottenere l'indice della colonna corrispondente a tale nome.

Il metodo "`dataProvider().nextFeature()`" applicato all'oggetto layer permette di recuperare, uno alla volta, il riferimento ad una geometria del layer selezionato. Il metodo "`attributeMap()`" applicato all'oggetto della geometria permette di recuperare il contenuto della tabella associata ad ogni geometria.

A questo punto è possibile effettuare il confronto tra gli attributi selezionati per il join. Nel caso il confronto risulta essere positivo, all'oggetto feature vuoto viene assegnata la nuova geometria e gli attributi di entrambi i layer attraverso i metodi "`setGeometry()`" e "`setAttributeMap()`".

A questo punto la feature completa viene aggiunta al file di output che è stato precedentemente creato con il costruttore "`QgsVectorFileWriter()`", il quale costruisce un file vettoriale vuoto contenete indicazioni sul tipo di geometrie che deve contenere (punti, linee o poligoni) e la lista degli attributi da associare ad ogni geometria.

Completata l'operazione il file creato viene aggiunto all'ambiente di lavoro con il metodo "`QgsMapLayerRegistry.instance().addMapLayer(NEW_LAYER)`".

Una volta ottenuto lo shapefile con l'insieme di dati desiderati, è possibile realizzare il grafico desiderato utilizzando il secondo tab dell'interfaccia, che mostrato in **Figura 4.15**.

Nel campo "Attributes" viene mostrata la lista delle proprietà del layer selezionato. In questo modo è possibile aggiungere alla lista sottostante gli attributi da visualizzare. Tale lista permette di selezionare anche il colore da associare nel disegno.

"Classification attribute" permette di selezionare quale attributo utilizzare per scalare la dimensione del grafico, mentre "size" permette di dimensionare il grafico in base al valore dell'attributo di classificazione selezionato.

In questo caso, purtroppo, la API Python non forniscono un accesso alla costruzione dell'*overlay* grafico (sopra ogni layer è presente uno strato di rivestimento, overlay appunto, per poter applicare etichette o grafici) perché allo stato attuale non è presente ancora un collegamento tra le API Python e le librerie C++.

Per superare questo limite è stato usato allo scopo il file di progetto prodotto da QuantumGIS. Tale file è un file XML, ovvero un file di testo con una struttura ad albero identificata tramite etichette. All'interno del file di progetto è identificata la composizione dell'area di overlay di ciascun layer del progetto.

Perciò l'applicazione provvede a:

- creare il file di progetto dello stato attuale dell'ambiente di lavoro;
- modificare il file di progetto, individuando il layer di interesse e aggiungendo le informazioni necessarie per costruire il grafico. Ovvero costruisce un albero XML avendo come elemento radice il layer oggetto dell'elaborazione;
- carica il file di progetto così modificato per visualizzare il risultato di tale operazione.

Utilizzando le API XML per Python si effettua la costruzione della parte di sotto-albero che permette di disegnare i grafici. L'operazione prevede la definizione di alcuni nodi. Il principale è il nodo "`overlay`" dove viene definito il tipo di disegno da realizzare e possiede una serie di nodi figli per definire:

- la dimensione che deve avere il grafico e su quale attributo scalare la dimensione (rispettivamente nodi "`renderer`" e "`scalingAttribute`");
- un nodo "`factory`" in cui è specificato il formato del diagramma. A sua volta, questo nodo, contiene come figli gli attributi, indicati con il numero di indice, che devono essere visualizzati nel grafico;
- ogni attributo ha, a sua volta, un nodo figlio in cui è specificato il colore da usare per la visualizzazione ("`brush`").

Il sotto-albero così costruito va appeso al nodo "`maplayer`" corrispondente al layer in oggetto.

Attraverso il metodo "`addProject()`" applicato al riferimento dell'applicazione (tale riferimento viene passato al plugin al momento dell'esecuzione quando viene invocato il metodo "`__init__()`" che fa partire il plugin) permette di ricaricare il file di progetto e visualizzare il risultato dell'elaborazione.

Con questo il processo di elaborazione è concluso. Nel capitolo successivo sono presentati alcuni esempi di casi d'uso che sono stati usati per elaborare la procedura più generale fin qui descritta. Tali esempi permetteranno di comprendere meglio sia

il lavoro svolto che le potenzialità dell'utilizzo di tali strumenti per facilitare la fusione di informazioni.

5 Casi d'uso reali

In questo capitolo sono presentati alcuni casi d'uso in cui è stato implementato l'approccio sviluppato per elaborare alcuni dei dati messi a disposizione dalle amministrazioni pubbliche. L'obiettivo è di mostrare i risultati ottenuti e di far comprendere il potenziale insito nei dati e in una loro corretta elaborazione. Anche attraverso l'unione con dati provenienti da sorgenti esterne.

Molte delle pubbliche amministrazioni italiane hanno iniziato da qualche tempo a rendere disponibili dati e informazioni relativi alla gestione della città e all'attività dell'amministrazione stessa. Ciò è iniziato sia da una spinta dal basso, dalla popolazione, che richiede, soprattutto in tempi di crisi economica, maggiore trasparenza sulle attività delle amministrazioni, sia in base anche a direttive europee, in cui si intravede, negli open data, una possibilità di sviluppo legata alla capacità di creare servizi a partire da queste informazioni.

5.1 I dati dei comuni di Bologna e Firenze

I comuni di Bologna [9] e Firenze [42] forniscono, in maniera pubblicamente accessibile e riutilizzabile, un ampio parco di dati relativi agli ambiti organizzativi della città di cui sono competenti. Questi due comuni hanno una dimensione paragonabile e hanno intrapreso entrambi, da qualche tempo, il processo di pubblicazione dei dataset in maniera abbastanza sistematica. Prendere in esame un confronto tra questi due comuni può essere significativo per migliorare la qualità del lavoro di ciascuno dei due e per individuare nuovi casi d'uso per gli stessi dati.

Tra i dati pubblicati sono presenti dati di tipo geografico relativi, ad esempio, sia ai confini dei vari quartieri o di aree verdi, sia stradari e posizione dei numeri civici. Altri dati sono relativi alla popolazione, come la distribuzione della popolazione tra le varie fasce di età o la dimensione delle famiglie, i dati sul reddito e i risultati delle elezioni, oltre, poi, i dati relativi all'amministrazione comunale come il bilancio comunale e le delibere.

I dati disponibili riguardano ambiti di vario tipo e sono disponibili in vari formati. In particolare, per i dati di nostro interesse, i formati disponibili sono SHP (shapefile) per i dati cartografici e CSV (comma-separated values) per le informazioni numeriche di tipo quantitativo.

Tra i due comuni in esame ci sono diverse differenze, sia sui dati scelti che il modo in cui sono aggregati. Una differenza importante è la granularità scelta per i dati. Prendendo in esame le informazioni relative alla popolazione dei due comuni, questi presentano diversa grana di tipo spaziale riguardo alla loro aggregazione. Ovvero sono suddivisi o per quartiere del comune, o per zona, o, a grana più fine, per sezione di censimento.

Mentre per il Comune di Firenze la granularità più fine è relativa ai quartieri, per il comune di Bologna esistono dati aggregati per aree più piccole dei quartieri. I dati sulla popolazione che invece provengono dai rilevamenti del censimento 2011 sono, per entrambi, relativi alle sezioni di censimento.

Figura 5.1: mappa di Bologna, le linee più sottili separano le zone, le più spesse separano i quartieri

Tra i file cartografici dei due comuni sono presenti file in cui sono rappresentate tutte le aree delle sezioni del censimento 2011 e ad ognuna di queste sezioni sono associate altre informazioni tra cui il quartiere e, nel caso di Bologna, la zona di cui fanno parte (dove per zona si intende un sottoinsieme del quartiere come mostrato in **Figura 1.1Figura 5.1**; per es. il quartiere Navile, nei documenti in cui i dati sono aggregati per zone, è diviso in tre zone: Bolognina, Lame e Corticella). In **Figura**

4.13 è mostrato in che modo sono in relazione Zone e Quartieri. Come si evince anche dalla cartina ogni Zona è completamente contenuta in un Quartiere.

Nel caso di Bologna, in genere i dati relativi a zone o quartieri sono presenti come serie storica in quanto fanno riferimento a dati dell'anagrafe del comune. Al contrario, i dati divisi per sezione di censimento sono relativi solo all'ultimo censimento (fine 2011).

Infine, i dati presenti fotografano sotto vari aspetti la composizione della popolazione. Vengono presi in esame: l'età anagrafica, lo stato civile, i luoghi di nascita (categorizzate in alcune macro aree geografiche che aumentano di dimensione con l'aumentare della distanza dal comune: comune di Bologna, provincia di Bologna, regione Emilia-Romagna, Italia, divisa in nord orientale, nord occidentale, centrale e meridionale e isole, e stati esteri), la dimensione delle famiglie; oltre a flussi migratori, natalità e decessi.

I dati possono, inoltre, divisi in due categorie. La prima è una fotografia dello stato della città e a questa categoria appartengono tutti i dati del censimento 2011 che mostrano, ad esempio, la composizione della popolazione organizzata per stato civile o età, oppure la dimensione delle famiglie.

La seconda è costituita da dati che mostrano la modifica dello stato interno tra un anno e l'altro. A questa seconda categoria fanno riferimento molti dei dati dell'anagrafe del comune; come il flusso di stranieri che prendono la residenza o si eliminano dai registri di residenza del comune, o, più in generale, nuovi iscritti e cancellati dall'anagrafe, oppure i dati su nuovi nati e sui decessi.

5.1.1 Nidi e scuole dell'infanzia

Un'analisi interessante può riguardare la situazione degli edifici scolastici rispetto a bambini e ragazzi in età scolare.

5.1.1.1 Asili nido

Tra i dati del comune di Bologna è presente un dataset con informazioni riguardanti gli asili nido del comune. A ciascuna scuola sono associate, tra le altre, informazioni relative al quartiere in cui è collocata e al numero di bambini in grado di accogliere.

A partire da questi dati, insieme ai dati sulla popolazione relativi alla fascia di età 0-2 anni aggregati per zona, si è cercato di individuare in che modo fossero distribuiti sul territorio comunale bambini e nidi.

La query è stata realizzata attraverso l'uso di due sotto-query di cui una estrae informazioni sul numero di bambini per quartiere e l'altra il numero di nidi e di posti disponibili per ciascun quartiere. Il risultato è unito secondo il nome del quartiere.

La seguente tabella mostra il risultato di tale elaborazione:

Quartiere	nidi_totali	bambini	iscritti
Borgo Panigale	4	711	164
Navile	19	1714	597
Porto	10	763	378
Reno	7	892	240
San Donato	11	761	322
Santo Stefano	10	1171	315
San Vitale	12	1204	400
Saragozza	6	813	268
Savena	12	1355	467

In **Figura 5.2** è mostrata una rappresentazione visuale del risultato in cui si mette in evidenza la differenza tra i bacini di utenza che gli asili nido dovrebbero servire.

La **Figura 5.3** mostra, invece, la percentuale di bambini che è riuscita ad accedere al servizio di asilo nido per ciascun quartiere.

5.1.1.2 Scuole dell'infanzia

Presso [40] è presente un database con accesso tramite endpoint SPARQL contenente informazioni su scuole italiane di ogni ordine e grado. Una informazione interessante è la presenza dell'indirizzo associato ad ogni scuola Anche se i dati, allo stato attuale, sembrano limitati, ovvero non sono indicizzate tutte le scuole, tali informazioni sono state utilizzate per localizzare le scuole sulla cartina.

Presso i portali dei comuni di Bologna e Firenze sono disponibili shapefile con l'elenco dei numeri civici della città con relative coordinate. L'elenco dei numeri

civici consiste in punti sulla mappa (quindi ad ogni punto è associata una coordinata) con collegati ad essi altre informazioni tra cui nome della via e numero civico.

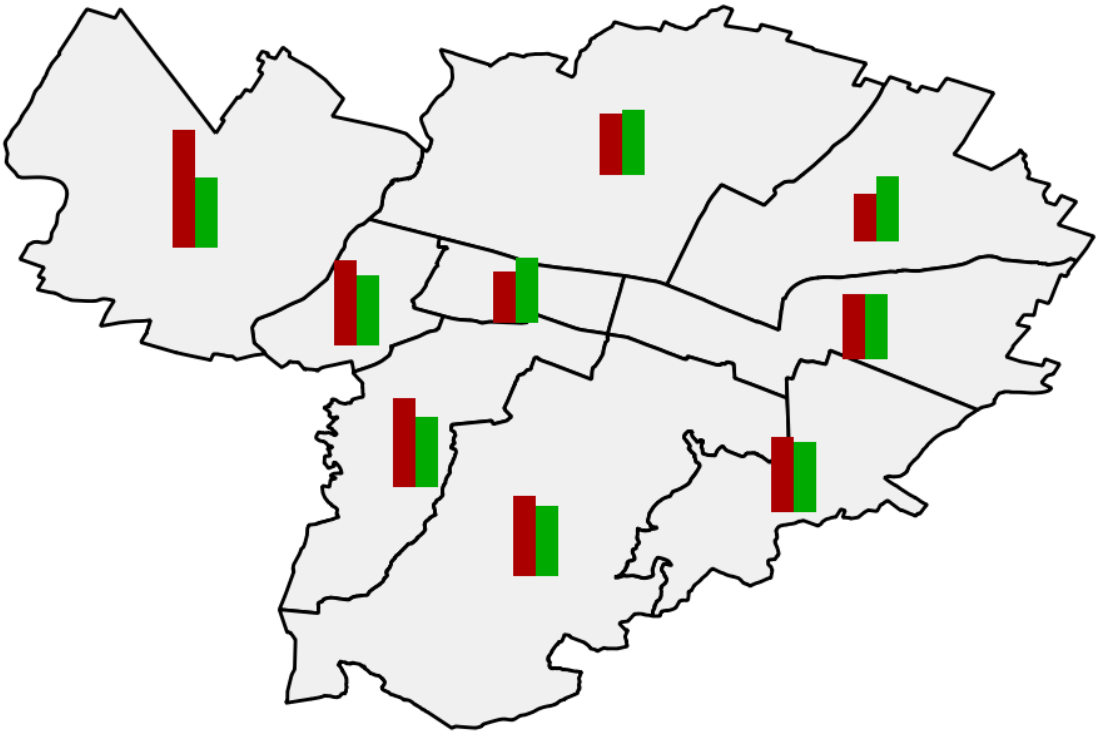

Figura 5.2: gli istogrammi indicano il rapporto tra bambini 0-2 anni residenti e numero di asili nido per quartiere (in rosso), confrontato con la media cittadina (in verde)

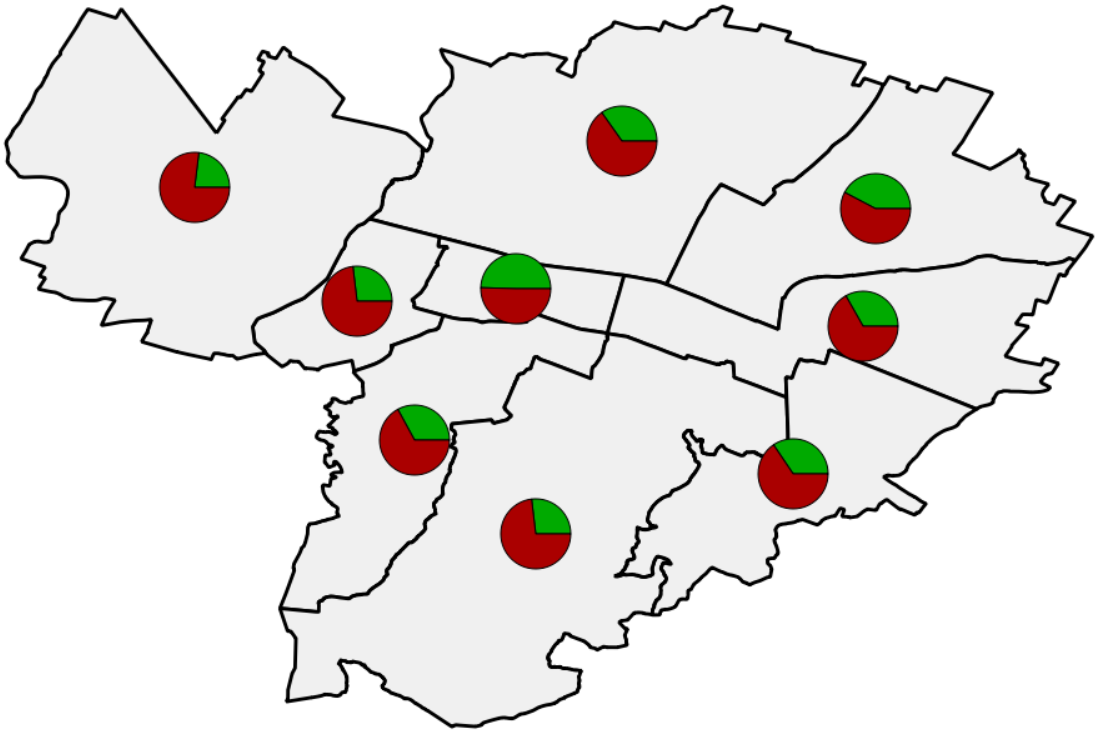

Figura 5.3: i digrammi circolari mostrano il numero di bambini 0-2 anni iscritti ai servizi di asilo nido nel quartiere (in verde) e i bambini non iscritti (in rosso)

Sarebbe possibile, in teoria, effettuare in un'unica query interrogazioni sia presso il servizio che mantiene le informazioni sulle scuole, sia presso l'endpoint messo a punto da noi con memorizzate le posizioni dei numeri civici della città ed effettuare un confronto testuale delle rispettive proprietà che contengono il nome della via e il numero civico.

codice	via	civico	quartiere	long	lat
BOAA003003	VIA XXI APRILE 1945	24	Saragozza	11.3150	44.4919
BOAA003014	VIA S.ISAIA	20	Saragozza	11.3337	44.4936
BOAA008006	VIA DANTE BENAZZI	3	Reno	11.286	44.4927
BOAA008006	VIA DANTE	3	Santo Stefano	11.3546	44.4869
BOAA01300N	VIA LAURA BASSI VERATTI	20	Santo Stefano	11.3681	44.4829
BOAA01301P	VIA GIUSEPPE MEZZOFANTI	32	Santo Stefano	11.3669	44.483
BOAA01302Q	VIA AUGUSTO MURRI	158	Santo Stefano	11.3687	44.4741
BOAA017001	VIA ETTORE NADALINI	1	Savena	11.3957	44.478
BOAA017012	VIA MILANO	13	Savena	11.3796	44.4726
BOAA017023	VIA DOMODOSSOLA	2	Savena	11.3873	44.4734
BOAA808005	VIA ADOLFO DE CAROLIS	23	Reno	11.299	44.4992
BOAA815008	VIA GIULIO VERNE	19	Navile	11.3614	44.5454
BOAA81502A	VIA RAFFAELE PETTAZZONI	1	Navile	11.3564	44.5424
BOAA816004	VIA ANTONIO DI VINCENZO	55	Navile	11.3455	44.5128
BOAA816015	VIA CRISTOFORO DA BOLOGNA	27	Navile	11.3396	44.5194
BOAA81700X	VIA ANGELO FINELLI	2	San Vitale	11.3491	44.5024
BOAA817011	VIA GIUSEPPE MASSARENTI	11	San Vitale	11.3617	44.4925
BOAA81800Q	VIA SCANDELLARA	54	San Vitale	11.3837	44.4969
BOAA81802T	VIA SCANDELLARA	54	San Vitale	11.3837	44.4969
BOAA852007	VIA LUIGI LONGO	4	Savena	11.3856	44.4677
BOAA852018	VIA DELL'ABBADIA	5	Porto	11.3341	44.4977
BOAA852018	VIA GIUSEPPE CESARE ABBA	5	Savena	11.3787	44.4651
BOAA853003	VIALE ALDO MORO	31	San Donato	11.3627	44.5084
BOAA853014	VIA DELL'ARTIGIANO	5	San Donato	11.3663	44.503
BOAA85400V	VIA FILIPPO BEROALDO	34	San Donato	11.3723	44.5005
BOAA85401X	VIA ALFREDO PANZINI	3	San Donato	11.3932	44509
BOAA854021	VIA ISABELLA ANDREINI	16	San Donato	11.3735	44.5039
BOAA854032	VIA FERRUCCIO BENINI	4	San Donato	11.3751	44.5047
BOAA85500P	VIA LORENZO BARTOLINI	2	Savena	11.382	44.4821
BOAA85501Q	VIA POPULONIA	7	Savena	11.3839	44.4881
BOAA85502R	VIA LORENZO BARTOLINI	4	Savena	11.3827	44.4819
BOAA87200C	VIA ALFONSO LOMBARDI	40	Navile	11.3517	44.5166
BOAA87201D	VIA DELLA DOZZA	8	Navile	11.3653	44.5295
BOAA87700G	VIA GALLIERA	74	Porto	11.3441	44.5034
BOAA87701L	VIA MILAZZO	3	Porto	11.3437	44.5031

Figura 5.4: output testuale dell'operazione di individuazione dei civici corrispondenti alla posizione delle scuole materne per la città di Bologna

Nome via e numero civico associati alle scuole sono memorizzati in un'unica proprietà contrariamente al dataset delle posizioni dei numeri civici, ed inoltre, non è usato un'unica forma per scrivere il nome della via (ad esempio Via San Donato viene spesso sostituita con Via S. Donato, oppure vie dedicate a persone possono essere scritte come nome e cognome o solo cognome). Per questi motivi allo stato attuale non è stato possibile eseguire l'interrogazione dei due servizi distinti in un'unica query dal momento che SPARQL non supporta la funzione di separazione dei termini all'interno di una stringa (*tokenizer*).

Per superare questo problema è stato necessario, una volta estratte le informazioni necessarie dall'endpoint SPARQL remoto, editare tale dataset in modo da permettere un confronto tra le stringhe attraverso regole espresse tramite espressioni regolari.

Combinando gli indirizzi delle scuole così modificati con le informazioni sulla posizione dei civici della città è stato possibile individuare la loro posizione su mappa attraverso l'uso di una query SPARQL. Tale query effettua un incrocio tra i dati estratti dai due dataset sulla base del valore delle stringhe degli indirizzi. In questo modo è possibile associare una coordinata spaziale ad una scuola.

Figura 5.5: posizione delle scuole di infanzia di Bologna

In **Figura 5.4** è mostrato l'output testuale ottenuto dall'elaborazione di questa query per la città di Bologna.

Di tale operazione ne è stato testato il funzionamento prendendo ad esempio le scuole materne di Bologna e Firenze. I risultati sono mostrati in **Figura 5.5** e **Figura 5.6**.

Figura 5.6: posizione delle scuole di infanzia di Firenze

5.1.2 Farmacie

Un approccio simile al precedente è stato affrontato per la posizione delle farmacie. I dati sulle farmacie di Bologna sono stati estratti da una pagina Web del servizio sanitario regionale dell'Emilia-Romagna [37]. I dati sono stati necessariamente editati prima di essere trasformati in RDF dato che non erano, in origine, in formato tabellare.

A partire da questi dati si è potuto, ad esempio, interrogare il sistema per ottenere la posizione di tutte le farmacie di uno specifico quartiere, come mostrato in **Figura 5.7**.

Per fare ciò è stato sufficiente specificare che i numeri civici da estrarre, e con cui fare il confronto con i nomi delle vie dei dati sulle farmacie, fossero appartenenti ad uno specifico quartiere.

A partire da questa query è stato possibile costruire una interrogazione più articolata in cui fosse effettuato un confronto con la popolazione residente. In questo modo si è calcolato il rapporto che intercorre tra il numero di residenti e le farmacie di uno specifico quartiere. Il risultato è mostrato in **Figura 5.8**.

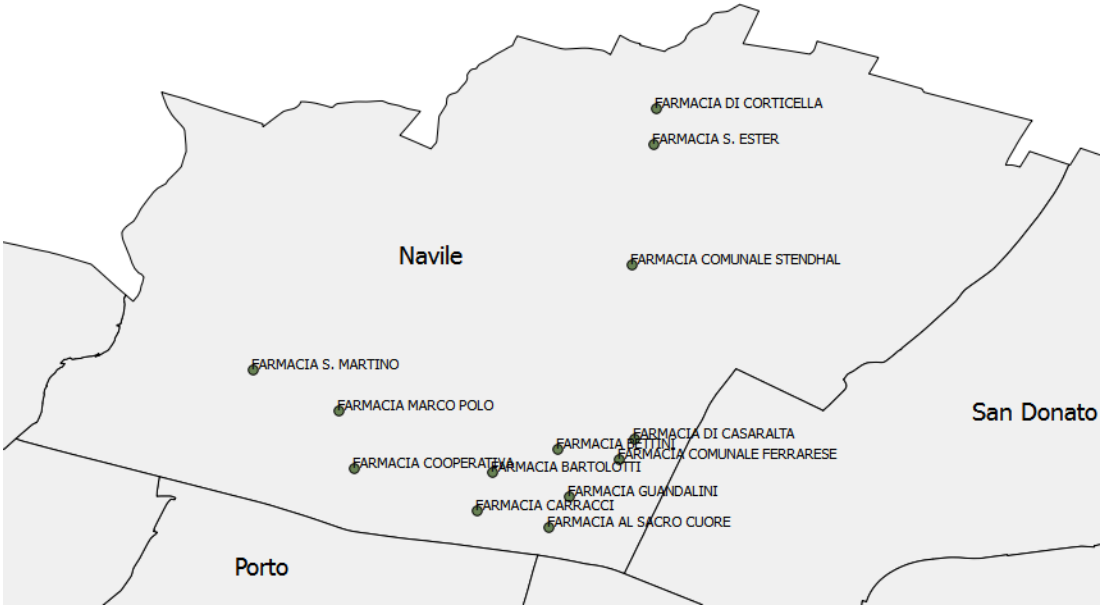

Figura 5.7: farmacie del quartiere Navile di Bologna

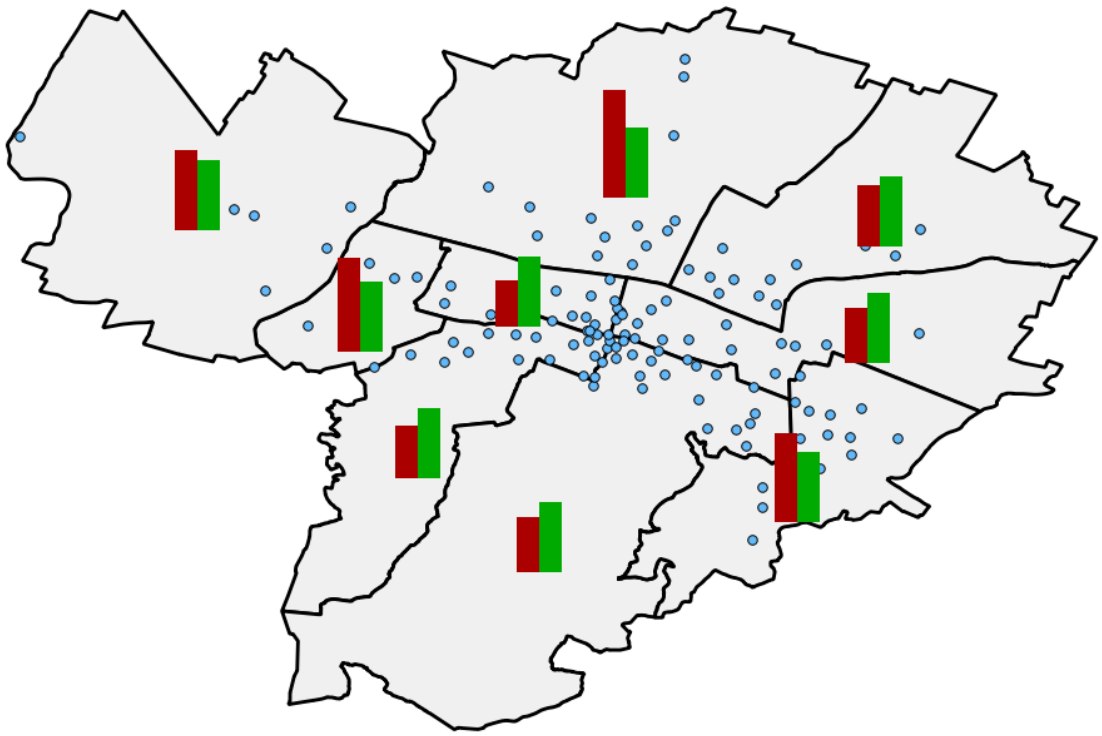

Figura 5.8: gli istogrammi indicano il rapporto tra residenti e numero di farmacie per quartiere (in rosso), confrontato con la media cittadina (in verde). I pallini indicano la posizione delle farmacie

Nel caso del comune di Firenze si è potuto fare un discorso simile in quanto i dati sulle posizioni delle farmacie erano già disponibili presso il portale OpenData del comune. I dati delle farmacie erano disponibili sotto forma di shapefile e tra le proprietà associate erano disponibili solo il nome e l'indirizzo.

Per portare a termine la query in oggetto è stato necessario associare ad ogni punto, indicate una farmacia, il quartiere in cui è posizionato. Per far ciò si è usato QuantumGIS che permette di effettuare un join spaziale. Tale operazione permette di creare un nuovo layer avendo come geometrie quelle di uno dei due layer e associando proprietà di una delle geometrie dei due layer in base ad una relazione di contenimento o intersezione tra le diverse geometrie.

Una volta fatto ciò si è potuto costruire una nuova query che riuscisse, così, a contare le farmacie presenti in ciascun quartiere ed ottenere un rapporto tra abitanti e farmacie per ogni quartiere della città.

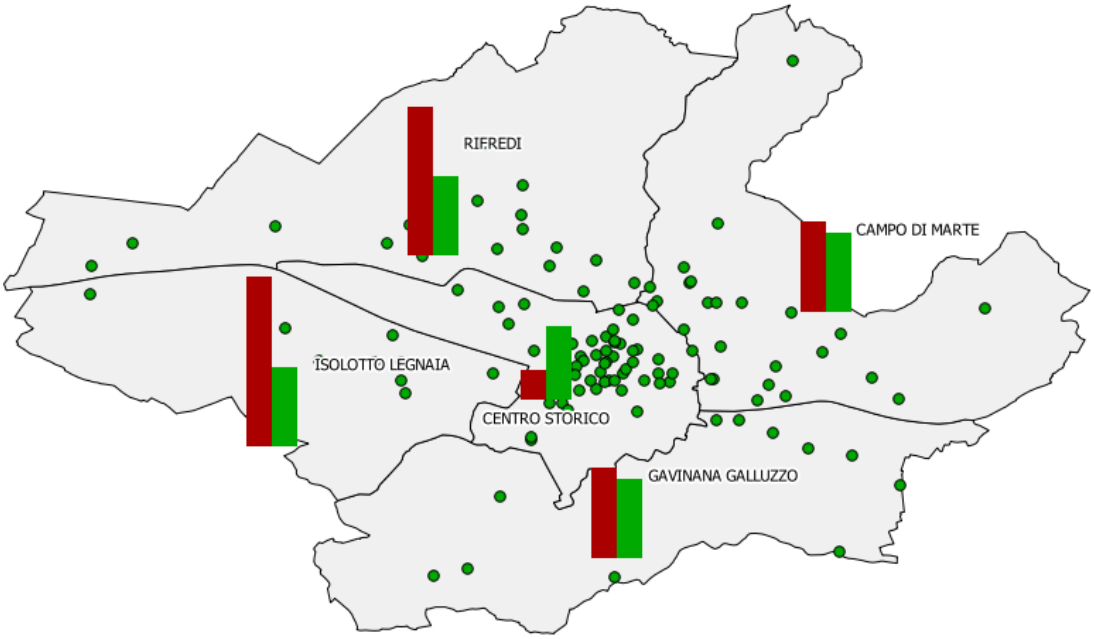

Figura 5.9: gli istogrammi indicano il rapporto tra residenti e numero di farmacie per quartiere (in rosso), confrontato con la media cittadina (in verde). I pallini indicano la posizione delle farmacie

5.2 Politiche 2013

In seguito alle recenti elezioni politiche, sono stati resi disponibili i risultati elettorali per le singole sezioni assieme all'affluenza alle urne. Attraverso i portali OpenData

dei diversi comuni italiani è possibile recuperare tali dati, e quindi anche usarli a scopo statistico.
All'interno di questo progetto si sono analizzati i risultati elettorali nei comuni di Bologna [9], Firenze [42] e Roma [43], e si sono confrontati con gli altri dati disponibili.

5.2.1 Dati sul voto e sulla popolazione

Come logico aspettarsi, essendo le modalità di voto simili in tutta Italia, i dataset su risultati elettorali e affluenza alle urne non avevano grosse differenze tra i diversi comuni presi in analisi. Per ogni sezione elettorale sono indicate le preferenze per ciascun partito, i votanti e le persone iscritte al voto che, nel caso di Bologna, erano differenziate per minori e maggiori di 25 anni.
La differenza principale consiste nei partiti iscritti al voto dal momento che la presentazione è indipendente per le diverse circoscrizioni elettorali. Altra differenza importante era la diversa granularità dei dati diversa per i diversi comuni.
I dati elettorali sono stati confrontati con alcune delle informazioni disponibili per la relativa città. In particolare, sono stati ipotizzati confronti, oltre che con i dai dell'affluenza, con le fasce di età 18-24 (naturalmente solo per i risultati della camera), maggiori di 65 anni e maggiori di 80 anni. Sono stai effettuati confronti anche con i dati sul reddito (questi ultimi dati sono disponibili solo per il comune di Bologna).

5.2.2 Interfaccia di interrogazione

Per permettere l'interrogazione di questi dati senza che l'utente conoscesse il linguaggio SPARQL o la struttura dei dati da interrogare, è stata creata una interfaccia che a partire da dei menù di selezione a tendina permette di creare una query da poter sottoporre all'endpoint SPARQL presso cui sono conservati i dati.
Per la costruzione di tali query sono state previste alcuni scheletri di query. La selezione dei parametri da parte dell'utente permettono di completare le parti mancanti ed ottenere il risultato desiderato.
In particolare, le query sono state parametrizzate per far si che si potesse scegliere dinamicamente il livello di aggregazione dei dati (zone, quartieri o intera città)

oppure se devono essere relativa a Camera dei Deputati o Senato della Repubblica. Inoltre è possibile combinare tali query attraverso l'incapsulamento in una query esterna per effettuare un confronto tra risultati elettorali e informazioni sulla popolazione.

L'interfaccia permette di ottenere dinamicamente la lista dei partiti attraverso una interrogazione al servizio trasparente all'utente. La risposta viene usata per popolare i menù a tendina che permettono di selezionare uno o più partiti di cui si vogliono ottenere informazioni sui risultati. Naturalmente tale risposta dipende sia dalla città che dalla camera selezionata.

Strutture di query realizzate:

- astensione: si individua la percentuale di partecipazione al voto per i diversi livelli di aggregazione a partire dal conteggio di iscritti e votanti nelle singole sezioni;
- redditi: a partire da dati sui redditi irpef e numero di contribuenti si individuano i valori medi di reddito per contribuente per l'area di aggregazione specificata (disponibile solo per il comune di Bologna);
- età 18-24: si individua la percentuale di popolazione in questa fascia di età rispetto al totale degli aventi diritto. Nel caso di Bologna è stato possibile ipotizzare, per via di una diversa composizione dei dati, il numero effettivo di votanti in questa fascia di popolazione e calcolarne la percentuale su votanti totali. Tale operazione è stata tentata dal momento che erano disponibili, a differenza degli altri comuni presi in analisi, informazioni sul numero di registrati al voto minori di 25 anni. Per questo è stato supposto che fosse possibile calcolare i votanti minori di 25 anni sottraendo il numero dei votanti al Senato al numero di votanti alla Camera;
- età 65+ e 80+: si individua la percentuale di popolazione in questa fascia di età rispetto al totale degli aventi diritto.

Ciascuna di queste query può essere eseguita sia indipendentemente, sia in modo da ottenere come risultato il confronto con la performance elettorale di uno o più partiti.

In seguito, il comune di Bologna ha rilasciato informazioni sulla popolazione aggregate per sezione elettorale, quindi con una granularità più fine. In questo modo è stato possibile aggiungere questo ulteriore livello di aggregazione. Ciò ha

comportato una modifica solo parziale degli scheletri di query già approntati permettendo di aggiungere tale possibilità all'interfaccia di interrogazione.

5.2.3 Risultati

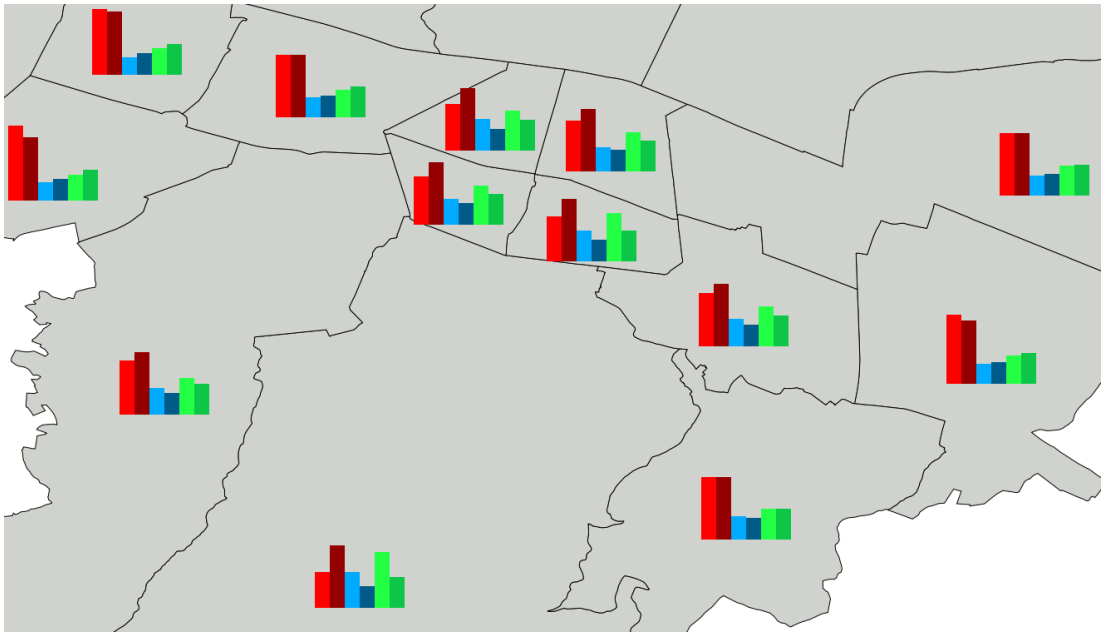

Figura 5.10: in figura sono visualizzati i risultati del PD (in rosso), del PDL (in blu) per le Politiche 2013 e il reddito medio (verde) a Bologna. La barra più chiara indicala media della singola zona, la più scura la media cittadina

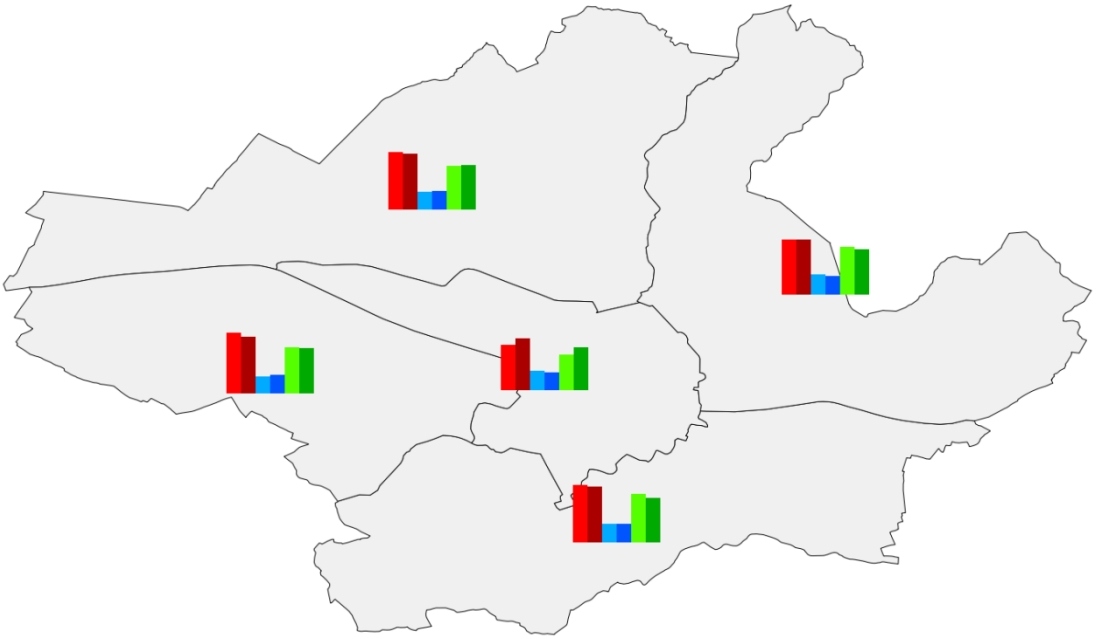

Figura 5.11: in figura sono visualizzati i risultati del PD (in rosso), del PDL (in blu) per le Politiche 2013 e la percentuale di ultrasessantacinquenni (verde) a Firenze. La barra più chiara indicala media della singola zona, la più scura la media cittadina

Dai risultati ottenuti dalla precedente elaborazione si è potuto ottenere una visualizzazione del risultato su cartina in grado di fornire un colpo d'occhio sui risultati migliore rispetto ad una semplice tabella.

Ad esempio, la **Figura 5.10** mostra i risultati dell'elaborazione circa la distribuzione del reddito tra le varie zone della città e l'andamento dei risultati del voto. In questo caso sono mostrati i risultati di due dei partiti maggiori ovvero Partito Democratico e Popolo della Libertà. Sono mostrati affiancati la media della zona e la media cittadina per evidenziare le differenze tra le diverse zone e rendere più immediata la lettura.

In **Figura 5.11** è mostrata, similmente all'esempio precedente, i risultati dell'elaborazione circa la presenza di ultrasessantacinquenni nei diversi quartieri della città di Firenze. I risultati sono affiancati alle prestazioni elettorali dei partiti Partito Democratico e Popolo della Libertà.

Discorso simile può essere svolto per gli altri confronti e per gli altri comuni analizzati. Per il comune di Roma non è stato possibile realizzare una visualizzazione su mappa dal momento che non è ancora disponibile una mappa vettoriale presso il portale OpenData del comune che permettesse di farlo.

6 Risultati sperimentali e valutazioni finali

Nei due precedenti capitoli sono state esposte e discusse le varie parti che compongono il progetto e alcuni esempi di risultati che si sono riusciti ad ottenere col processo fin qui sviluppato. In questo capitolo verranno esposti i risultati di alcuni test condotti per cercare di valutare in che modo la complessità di una query possa influenzare la durata dell'esecuzione.
La valutazione dei tempi di risposta può essere determinate per determinare il dimensionamento del sistema, affinché lo stesso sia in grado di gestire le richieste previste. In questo capitolo, quindi, è stato affrontato il problema di determinare quanto una query SPARQL utilizza delle risorse a disposizione e in che modo variano le risorse utilizzate in funzione di vari parametri. Ciò è stato determinato valutando il tempo di risposta impiegato per eseguire una singola query.

6.1 Test preliminari

Tutti i test sono stati eseguiti su una macchina con processore Intel i5-3210M @2,50 GHz con 4GB di RAM, configurata con sistema Windows 7 64bit e Virtuoso Opensource 64bit.
Per valutare la velocità di risposta sono state utilizzate query con diversa complessità e che, in fase di esecuzione, accedono a dataset di diverse dimensioni.
Il rilevamento dei risultati è stato effettuato modificando l'interfaccia di interrogazione realizzata, in modo da contare il tempo trascorso tra l'invio della richiesta e la ricezione della risposta.
Java mette a disposizione il metodo statico "`System.currentTimeMillis()`" che permette di recuperare il valore corrente dell'orologio in millisecondi. In questo modo è possibile calcolare il tempo trascorso tra i due istanti temporali che ci interessano.

6.1.1 Composizione delle query e risultati

Le query scelte sono state classificate e determinate in base ai seguenti parametri:

- dimensione dell'input: a seconda della dimensione dell'input il tempo di esecuzione può essere differente perché potrebbe essere necessario più tempo per completare tutte le operazioni di *graph match*;
- dimensione dell'output: il numero di triple in output coinvolte nel graph match può in qualche modo far variare la durata dell'esecuzione;
- dimensione del pattern matching: se e come il numero di triple coinvolte in un unico pattern match influenza l'esecuzione;
- numero di sotto-query coinvolte: la presenza di sotto-query può far sì che i tempi di esecuzione si dilatino;
- filtro dell'output e operazioni di aggregazione dei risultati: come effettuare una selezione dell'output in base al confronto.

I dataset utilizzati hanno una dimensione non molto variabile tra di loro dato che si riferiscono, in un certo qual modo, a dati prodotti a partire da una sorgente di informazione con numeri non molto variabili. Ovvero si tratta di dati prodotti a partire da una popolazione limitata e aggregati con un livello di granularità non sensibilmente differente per cui le operazione che hanno coinvolto un graph match limitato hanno prodotto un risultato in un tempo molto limitato e al di sotto dei 100 millisecondi. Al contrario, operazioni in cui erano coinvolte delle sotto-query hanno comportato un tempo di esecuzione maggiore.

Un gruppo di query utilizzate è stato quello relativo al risultato dell'astensionismo nel comune di Bologna. Il dataset iniziale è composto da dati aggregati per sezione. Sono state confrontate diverse query con caratteristiche diverse:

- Query_1B: estrazione in output di alcune proprietà senza elaborazione (**Figura 6.2**);
- Query_2B: estrazione degli stessi dati con aggregazione sulla base di una delle proprietà portate in output;
- Query_3B: come Query_2B ma con una aggregazione diversa tale da portare in output un numero minore di risultati;

L'esecuzione di queste query non ha mostrato nessuna sostanziale differenza. Tutte sono state completate in media in un tempo pari a circa 90 – 100 ms. Query_1 è sembrata avere un tempo medio, calcolato sulla base di 20 esecuzioni, di pochi millisecondi in più (circa 5 ms) probabilmente perché l'output in uscita era maggiore

delle altre, comunque non è una differenza significativa tale da poter portare a conclusioni certe.

```
PREFIX ris:<http://dati.comune.bologna.it/Politiche_2013/Risultati_Camera_Deputati#>
PREFIX aff:<http://dati.comune.bologna.it/Politiche_2013/Affluenza_Camera_Deputati#>
PREFIX qr:<http://dati.comune.bologna.it/Quartieri#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>

SELECT (?num AS ?num_sez_el)
        ?voti_par1 (xsd:float((?voti_par1))/(?vv)*100 AS ?perc_par1_zona)
WHERE{
       ?sez ris:Sezione ?num;.
       ?sez ris:P.d.Voti ?voti_par1.
       ?a_sez aff:Sezione ?num;
              aff:VotiValidi ?vv.
}
```

Figura 6.1: Query_1A

```
PREFIX qr:<http://dati.comune.bologna.it/Quartieri#>
PREFIX af:<http://dati.comune.bologna.it/Politiche_2013/Affluenza_Camera_Deputati#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>

SELECT ?num_sez_el (?el_m_min+?el_f_min+?el_m_mag+?el_f_mag-?votm-?votf AS ?astenuti_sez)
       (?el_m_min+?el_f_min+?el_m_mag+?el_f_mag AS ?aventi_diritto_sez)
       ((xsd:float(?el_m_min)+?el_f_min+?el_m_mag+?el_f_mag-?votm-
                    ?votf)*100/(?el_m_min+?el_f_min+?el_m_mag+?el_f_mag) AS ?perc_sez)
WHERE{
       ?sez af:Sezione ?num_sez_el;
              af:VotantiMas ?votm;
              af:VotantiFem ?votf;
              af:MasMin25 ?el_m_min;
              af:FemMin25 ?el_f_min;
              af:MasMag25 ?el_m_mag;
              af:FemMag25 ?el_f_mag.
       FILTER (?num_sez_el < 438)
}
ORDER BY ?num_sez_el
```

Figura 6.2: Query_1B

Allo stesso modo si è comportato a riguardo del calcolo dei risultati relativi ad un singolo partito. Il tempo di esecuzione è stato intorno ai 40 ms con una tendenza ad impiegare più tempo nel caso di una dimensione dell'output maggiore.

Queste query sono costruite in modo simile alle precedenti ed in seguito saranno indicate come Query_xA (con x ad indicare un numero diverso a seconda dell'aggregazione dei risultati). Vedere **Figura 6.1**.

```
PREFIX ris:<http://dati.comune.bologna.it/Politiche_2013/Risultati_Camera_Deputati#>
PREFIX aff:<http://dati.comune.bologna.it/Politiche_2013/Affluenza_Camera_Deputati#>
PREFIX qr:<http://dati.comune.bologna.it/Quartieri#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>

SELECT *
WHERE{
      {
      SELECT (?num AS ?num_sez_el)
            ?voti_par1 (xsd:float((?voti_par1))/(?vv)*100 AS ?perc_par1_zona)
      WHERE{
            ?sez ris:Sezione ?num;.
            ?sez ris:P.d.Voti ?voti_par1.
            ?a_sez aff:Sezione ?num;
                  aff:VotiValidi ?vv.
            }
      }
      {
      SELECT ?num_sez_el (?el_m_min+?el_f_min+?el_m_mag+?el_f_mag-?votm-?votf
                                          AS ?astenuti_sez)
            (?el_m_min+?el_f_min+?el_m_mag+?el_f_mag AS ?aventi_diritto_sez)
            ((xsd:float(?el_m_min)+?el_f_min+?el_m_mag+?el_f_mag-?votm-?votf)*
                        100/(?el_m_min+?el_f_min+?el_m_mag+?el_f_mag) AS ?perc_sez)
      WHERE{
            ?sez aff:Sezione ?num_sez_el;
                 aff:VotantiMas ?votm;
                 aff:VotantiFem ?votf;
                 aff:MasMin25 ?el_m_min;
                 aff:FemMin25 ?el_f_min;
                 aff:MasMag25 ?el_m_mag;
                 aff:FemMag25 ?el_f_mag.
            FILTER (?num_sez_el < 438)
            }
      ORDER BY ?num_sez_el
      }
}
```

Figura 6.3: Query_1A+B

Le stesse query sono state usate come sotto-query con lo scopo di ottenere in un'unica tabella entrambi i gruppi di risultati per facilitarne il confronto.

In quest'ultimo caso il tempo di esecuzione è aumentato, come prevedibile, assestandosi in media intorno ai 140 ms escluso il caso in cui le query non prevedevano una aggregazione dove il tempo di elaborazione si è assestato intorno ai 700 ms. Dal momento che l'output di entrambe le sotto-query era sensibilmente più ampio rispetto agli altri casi e che, quindi, risulta essere necessario più tempo

incrociare i due output, incrocio effettuato in base al binding di una variabile comune alle due query.

query	1	2	3
A	40 ms	38 ms	38 ms
B	95 ms	90 ms	90 ms
A+B	700 ms	140 ms	140 ms

Figura 6.4: tempi di esecuzione delle query su partiti e astensione

In **Figura 6.4** è mostrata la tabella riassuntiva di questi test. I numeri in alto indicano il livello aggregazione e le lettere a lato la tipologia della query. Con A+B si intende la query formata usando Query_xA e Query_xB come sotto-query.

Un altro caso preso in esame è quello delle posizioni delle farmacie. Come descritto in precedenza, la soluzione si basa sul confronto tra le stringhe in cui sono contenuti gli indirizzi dei punti indicanti la posizione dei civici (indicata in seguito come Query_A e mostrata in **Figura 6.5**), e delle farmacie (indicata come Query_B e mostrata in **Figura 6.6**). L'interrogazione è ottenuta combinando due sotto-query (**Figura 6.7**) che effettuano l'estrazione dei dati, in seguito viene effettuato il filtro dei risultati del prodotto cartesiano delle due sotto-query in base ai valori degli indirizzi.

```
PREFIX civ:<http://dati.comune.bologna.it/civici_shp#>

SELECT ?via ?civico ?long ?lat
WHERE{
       ?s civ:Quartiere ?q;#"Navile"^^<http://www.w3.org/2001/XMLSchema#string>;
          civ:Civico ?civico;
          civ:Ncivsub ?civico;
          civ:Denominazi ?via;
          civ:X ?long;
          civ:Y ?lat .
}
```

Figura 6.5: Query_A

In questo caso il tempo di esecuzione combinata delle due query risulta essere minore delle due sotto-query eseguite singolarmente. I risultati sono mostrati in **Figura 6.8**.

```
PREFIX fr:<http://www.salute.bologna.it/Elenco_farmacie_Bologna#>

SELECT ?farm ?viaf ?num ?f
WHERE{
       ?f fr:NomeFarmacia ?farm ;
          fr:Indirizzo ?viaf;
          fr:Numero ?num .}
}
```

Figura 6.6: Query_B

```
PREFIX fr:<http://www.salute.bologna.it/Elenco_farmacie_Bologna#>
PREFIX civ:<http://dati.comune.bologna.it/civici_shp#>

SELECT (?farm AS ?Nome_Farmacia) ?via ?civico ?long ?lat
WHERE{
       {
       SELECT ?via ?civico ?long ?lat
       WHERE{
              ?s civ:Civico ?civico;
                 civ:Ncivsub ?civico;
                 civ:Denominazi ?via;
                 civ:X ?long;
                 civ:Y ?lat .
              }
       }
       {
       SELECT ?farm ?viaf ?num ?f
       WHERE{
              ?f fr:NomeFarmacia ?farm ;
                 fr:Indirizzo ?viaf;
                 fr:Numero ?num .
              }
       }
FILTER(regex(?via, SUBSTR(?viaf,4), "i"))
FILTER(?civico=?num)}
```

Figura 6.7: Query_A+B

Query_A	Query_B	A+B
750 ms	16 ms	590 ms

Figura 6.8: tempi di esecuzione del calcolo della posizione delle farmacie

Da questi risultati si può dedurre che una parte importante dell'esecuzione dipenda dalla dimensione dell'output portato in uscita, anche se in uno stadio intermedio dell'esecuzione si opera con informazioni di grosse dimensioni. Infatti la Query_A

ha un numero di risultato molto più ampio della seconda e della combinazione delle due, ovvero un risultato per ogni civico di Bologna.

```
PREFIX fr:<http://www.salute.bologna.it/Elenco_farmacie_Bologna#>
PREFIX civ:<http://dati.comune.bologna.it/civici_shp#>
PREFIX res:<http://dati.comune.bologna.it/Residenti_1986-2012#>
PREFIX qr:<http://dati.comune.bologna.it/Quartieri#>

SELECT ?qn ?tot_farmacie ?totale (?totale/?tot_farmacie AS ?rapporto)
WHERE{
      {SELECT ?qn (SUM(?totz) AS ?totale
      WHERE{
            ?idr res:2011 ?totz; res:Zone ?o .
            ?id qr:Zona_nome ?o;
                qr:Quar_nome ?qn .
            }
      GROUP BY ?qn
      }
      {SELECT (COUNT(?farm) AS ?tot_farmacie) ?quartiere
      WHERE{
            {SELECT ?quartiere ?via ?civico
            WHERE{
                  ?s civ:Quartiere ?quartiere;
                     civ:Civico ?civico;
                     civ:Ncivsub ?civico;
                     civ:Denominazi ?via;
                     civ:X ?long;
                     civ:Y ?lat .}}
            {SELECT ?farm ?viaf ?num ?f
            WHERE{
                  ?f fr:NomeFarmacia ?farm ;
                     fr:Indirizzo ?viaf;
                     fr:Numero ?num .
                  }
            }
            FILTER(regex(?via, SUBSTR(?viaf,4), "i"))
            FILTER(?civico=?num)
      }
      GROUP BY ?quartiere}
      FILTER(regex(?qn, ?quartiere, "i"))
}
```

Figura 6.9: Query_A+B+C

Un ulteriore test è stato effettuato a partire dalla combinazione delle query Query_A e Query_B, appena descritte, combinate insieme come unica query (ovvero A+B con i risultati mostrati in **Figura 6.8**) e facente parte di una query più generale (mostrata in **Figura 6.9**) in cui è presente un'altra sotto-query, che indicheremo come Query_C

(**Figura 6.10**), in cui vengono estratte informazioni relative al numero di abitanti in un quartiere. L'output finale consiste nel calcolare il numero di persone medio per quartiere a cui una farmacia deve prestare servizio, e il cui risultato finale è stato già mostrato in **Figura 5.8**. In **Figura 6.11** è mostrata la tabella riassuntiva dei tempi di esecuzione. In questo caso la combinazione QueryA + Query_B ha una durata maggiore perché esegue in più una operazione di aggregazione delle farmacia in base al quartiere per poterne calcolare il numero totale per la singola zona.

```
PREFIX fr:<http://www.salute.bologna.it/Elenco_farmacie_Bologna#>
PREFIX civ:<http://dati.comune.bologna.it/civici_shp#>
PREFIX res:<http://dati.comune.bologna.it/Residenti_1986-2012#>
PREFIX qr:<http://dati.comune.bologna.it/Quartieri#>

SELECT ?qn (SUM(?totz) AS ?totale
WHERE{
       ?idr res:2011 ?totz; res:Zone ?o .
       ?id qr:Zona_nome ?o;
           qr:Quar_nome ?qn .
}
GROUP BY ?qn
```

Figura 6.10: Query_C

A+B	Query_C	A+B+C
650 ms	10 ms	4950 ms

Figura 6.11: tempi di esecuzione per calcolo dei rapporti tra abitanti e farmacie per quartiere

In questo caso il tempo impiegato non sembra avere una sequenza logica particolare. Dal momento che gli output delle query intermedie e finali sono anche limitati in dimensione, il tempo di esecuzione molto alto potrebbe dipendere dal modo in cui viene gestita e costruita internamente al sistema la query che interroga il database relazionale ad oggetti su cui è basato Virtuoso.

6.1.2 Test di carico

In seguito sono stati condotti anche dei test per verificare il comportamento del sistema di interrogazione, ed in particolare di Virtuoso Opensource, in situazioni di più richieste contemporanee.

Il test è stato condotto modificando l'interfaccia di interrogazione permettendo di istanziare più thread in grado di eseguire la richiesta. In questo caso non c'è stata

alcuna visualizzazione del risultato e il conteggio del tempo è stato relativo solo alla durata della fase di invio della richiesta, esecuzione e restituzione dei risultati.

	1	5	10	20	40
Query_A	140 ms	350 ms	545 ms	952 ms	1470 ms
Query_B	700 ms	2205 ms	4185 ms	6800 ms	12325 ms
Query_C	950 ms	2352 ms	4725 ms	9778 ms	14587 ms

Figura 6.12: test di carico: risultati sperimentali

Per questo test sono state scelte tre query di diversa complessità e composizione e soprattutto con tempi di esecuzione, nel caso singolo, abbastanza diversi.
In **Figura 6.12** e **Figura 6.13** sono mostrati i risultati di tale esecuzione.

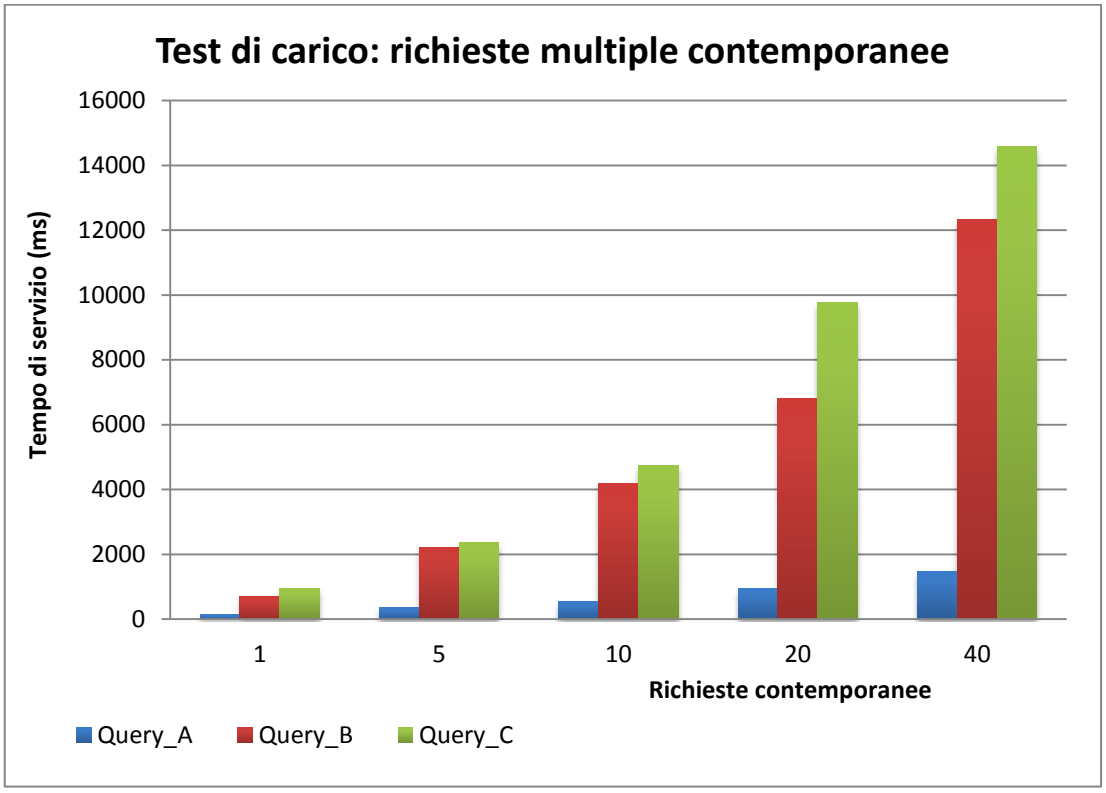

Figura 6.13: grafico relativo ai risultati mostrati in **Figura 6.12**

I dati si riferiscono alla durata media di servizio per ogni singola richiesta. Le richieste sono attivate in contemporanea all'avvio de test e viene portato in output la durata della specifica richiesta. Ogni esecuzione è stata ripetuta 5 volte, per limitare le fluttuazioni dei risultati dovuti alle interferenze degli altri processi in esecuzione sulla macchina, ed in tabella è riportato il valore medio delle 5 diverse esecuzioni.

Il grafico mette in evidenza come l'esecuzione di più richieste in contemporanea fa aumentare il tempo medio di servizio in maniera non lineare con il numero di richieste. Il grafico mostra che raddoppiare le richieste non fa raddoppiare il tempo di servizio. Infatti, Virtuoso opera con un sistema a pool di thread con un valore di default pari a 25. Questa scelta realizzativa fa si che, durante l'esecuzione di un numero di richieste maggiore dei thread disponibili, le richieste eccedenti restino in attesa prima di essere gestite. Questo comportamento è risultato evidente durante l'esecuzione del test con 40 richieste contemporanee. Durante questa prova si è creata una distinzione abbastanza netta tra i tempi di esecuzione delle richieste gestite per prime e le richieste rimaste in attesa.

6.2 Sviluppi futuri

La soluzione proposta, seppure ancora in fase prototipale, può essere usata per procedere attivamente alla trasformazione dei dataset disponibili in formato RDF. La relativa facilità mostrata nella possibilità di integrare dati progettati in origine senza una struttura comune è sicuramente un punto di forza nelle capacità offerte dalle tecnologie utilizzate.

Allo stato attuale mancano ancora importanti integrazioni che potrebbero portare ulteriori vantaggi nell'utilizzo dei dati e potrebbero facilitare ulteriormente l'integrazione e la derivazione di una informazione. L'utilizzo di una ontologia scelta tra quelle più diffuse o la creazione di una ontologia ad hoc per individuare concetti comuni all'interno dei dataset proposti dalle amministrazioni può essere utile per effettuare confronti con maggiore semplicità sfruttando i concetti più generali per demandare al ragionatore calcoli ed estrazioni di dati specifici dai dataset.

In quest'ottica può essere ipotizzata una estensione per Any23 in cui venga proposto all'utente, durante la fase di trasformazione, di promuovere ad URI alcune delle proprietà. Si potrebbe anche ipotizzare un sistema che proponga all'utente una scelta tra i concetti conosciuti provando a indovinare a partire dal nome della risorsa.

Per quanto riguarda la fase di interrogazione, una operazione successiva potrebbe essere la creazione di una interfaccia di aiuto alla creazione di query che operi in base alla struttura dei dataset sottostanti. Attraverso una interrogazione continua al

motore di query potrebbe proporre soluzioni di interrogazione e auto-completamento. Inoltre, la visualizzazione della struttura dei dati può essere di aiuto nella creazione senza dover impiegare tempo, in precedenza, ad esplorare i grafi presenti.

Anche l'estensione per QuantimGIS potrebbe essere oggetto di modifiche ed estensioni permettendo l'integrazione e visualizzazione di dati in modo più versatile.

Conclusioni

I promotori dell'open government sostengono che esistono molti vantaggi per governi e amministrazioni nella condivisione delle grandi quantità di informazioni che collezionano. Questi benefici vanno da un governo più trasparente, alla creazione di partnership pubblico-privato che possono portare innovazione e sviluppo al di fuori del governo e, con essi, migliorare la fornitura di servizi.

In anni recenti, gli Open Government Data (OGD) stanno emergendo come canale di comunicazione vitale tra l'amministrazione e i sui cittadini. Sono, inoltre, stati sviluppati numerosi portali a livello nazionale e internazionale per distribuire dataset OGD. Questi dati comprendono un ampio spettro di informazioni significative della vita di tutti i giorni.

Per le amministrazioni, il costo di rilasciare i dati attraverso questi portali è minore di pubblicarli attraverso applicazioni o dentro report. Ma per gli utilizzatori ci possono essere problemi di usabilità e di interoperabilità dei dati. Molto spesso infatti i dati vengono pubblicati grezzi, in strutture e formati eterogenei che rendono necessario un lavoro di pulizia prima di poter essere processati automaticamente da una macchina.

Per accelerare e facilitare l'uso di tali dati risulta necessaria una infrastruttura in grado di semplificare tale lavoro.

Il nostro obiettivo è stato quello di esplorare la fase di rilascio e pubblicazione dei dati di alcune pubbliche amministrazioni.

Partendo dall'analisi degli esperimenti di Data.gov e Data.gov.uk è stato individuato un processo di rilascio e pubblicazione dei dati in formato RDF tale da consentire con relativa facilità il recupero di informazioni e la visualizzazione dei risultati sotto forma di grafici.

Il percorso seguito è partito da una individuazione e scelta di dati, tra quelli disponibili a livello nazionale, che potessero essere usati da esempio per la realizzazione di casi d'uso di interesse per la pubblica amministrazione. Si è affrontato il problema dell'adattamento dei dataset disponibili affinché fosse possibile processarli attraverso strumenti automatici per trasformarli in formato RDF.

In seguito si è approntato un sistema che costruisse, a partire da una struttura di base e in base alle scelte dell'utente, delle query in grado di estrarre informazioni dai dati messi a disposizione delle pubbliche amministrazioni prese in esame. Infine, è stata approntata una estensione per QuantumGIS che facilitasse l'operazione di visualizzazione su mappa di risultati e statistiche elaborati a partire dai dati legati ad aree specifiche del territorio permettendo delle personalizzazioni in base alle esigenze dell'utente.

Nella fase finale sono stati effettuati alcuni test sulle prestazioni di Virtuoso Opensource nel gestire le richieste. Tali test hanno mostrato una buona performance nonostante siano stati eseguiti su un dispositivo dalle capacità di calcolo limitate.

Il lavoro svolto ha dimostrato l'utilità e l'ampia riusabilità del supporto realizzato; infatti, oltre a rendere quasi immediata la generazione di query, relative al caso di studio delle elezioni politiche 2013, ad utenti poco esperti, è stato anche possibile adattare, in modo relativamente semplice, il processo sviluppato alle diverse situazioni riscontrate tra i diversi portali utilizzati per reperire gli Open Data.

Rimangono però aperte diverse possibili direzioni di sviluppo futuro. Da una parte, è possibile ipotizzare un processo di standardizzazione dell'operazione di conversione, in cui i dataset prodotti sono catalogati e convertiti. Prima identificandoli con identificatori unici, ovvero la creazione di snapshot dei dati in uno specifico istante temporale, e in seguito la creazione di uno strato di conversione che permetta la promozione di proprietà ad URI definiti all'interno di ontologie diffuse, oppure la creazione di una ontologia specifica che possa essere, in seguito, collegata ad altri concetti definiti esternamente. Dall'altra, si ritiene importante mettere a punto nuove interfacce di interrogazione che consentano agli utenti una maggiore libertà di manovra nella creazione di query, seppure a discapito della semplicità d'uso.

Bibliografia

[1] W3C Technical Architecture Group, «Architecture of the World Wide Web, Volume One,» 2004. [Online]. Available: http://www.w3.org/TR/2004/REC-webarch-20041215/.

[2] O. Lassila e R. R. Swick, «Resource Description Framework (RDF) Model and Syntax Specification,» W3C, 1999. [Online]. Available: http://www.w3.org/TR/1999/PR-rdf-syntax-19990105/.

[3] W3C, «W3C Semantic Web Activity,» 2013. [Online]. Available: http://www.w3.org/2001/sw/.

[4] W3C, «Linked Data,» 2013. [Online]. Available: http://www.w3.org/standards/semanticweb/data.

[5] T. Berners-Lee, «Linked Data,» 2006. [Online]. Available: http://www.w3.org/DesignIssues/LinkedData.html.

[6] T. Heath, «Linked Data - Connect Distributed Data across the Web,» [Online]. Available: http://linkeddata.org/. [Consultato il giorno 2013].

[7] W3C, «Vocabularies,» 2013. [Online]. Available: http://www.w3.org/standards/semanticweb/ontology.

[8] «Creative Commons,» 2013. [Online]. Available: http://creativecommons.org/.

[9] Comune di Bologna, «OpenData Bologna,» 2013. [Online]. Available: http://dati.comune.bologna.it/.

[10] E. Kalampokis, E. Tambouris e K. Tarabanis , «A classification scheme for open government data: towards linking decentralised data,» *Int. J. Web Engineering and Technology*, vol. 6, n. 3, 2011.

[11] S. Nigel, K. O'Hara, T. Berners-Lee, N. Gibbins, H. Glaser, W. Hall e M. Schraefel, «Linked Open Government Data: Lessons from Data.gov.uk,» *IEEE INTELLIGENT SYSTEMS,* Maggio/Giugno 2012.

[12] Y. Shafranovich, «RFC: 4180 - Common Format and MIME Type for

Comma-Separated Values (CSV) Files,» Ottobre 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4180.

[13] J. Hendler, J. Holm, C. Musialek e G. Thomas, «US Government Linked Open Data: Semantic.data.gov,» *IEEE INTELLIGENT SYSTEMS,* Maggio/Giugno 2012.

[14] Rensselaer Polytechnic Institute, «Tetherless World Constellation,» 2013. [Online]. Available: http://tw.rpi.edu/.

[15] L. Ding, T. Lebo, J. S. Erickson, D. DiFranzo, G. T. Williams, X. Li e J. Michaelis, «TWC LOGD: A Portal for Linked Open Government Data Ecosystems,» *J. Web Semantics,* vol. 9, n. 3, 2011.

[16] T. Berners-Lee, «URI References: Fragment Identifiers on URIs,» Aprile 1997. [Online]. Available: http://www.w3.org/DesignIssues/Fragment.html.

[17] B. DuCharme, in *Learning SPARQL*, O'Reilly, 2011.

[18] D. Brickley e R. Guha, «RDF Vocabulary Description Language 1.0: RDF Schema,» W3C, 2013. [Online]. Available: http://www.w3.org/TR/rdf-schema/.

[19] D. Brickley e L. Miller, «FOAF Vocabulary Specification 0.98,» Agosto 2010. [Online]. Available: http://xmlns.com/foaf/spec/.

[20] S. Perreault, «RFC: 6351 - xCard: vCard XML Representation,» Internet Engineering Task Force, Agosto 2011. [Online]. Available: http://tools.ietf.org/html/rfc6351.

[21] R. Iannella e J. McKinney, «vCard Ontology,» W3C, Maggio 2013. [Online]. Available: http://www.w3.org/TR/vcard-rdf/.

[22] DCMI, 2013. [Online]. Available: http://dublincore.org/.

[23] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider e L. A. Stein, «OWL Web Ontology Language,» W3C, 2004. [Online]. Available: http://www.w3.org/TR/owl-ref/.

[24] S. Harris e A. Seaborne, «SPARQL 1.1 Query Language,» W3C, 2012. [Online]. Available: http://www.w3.org/TR/2012/WD-sparql11-query-20120724/.

[25] E. Prud'hommeaux, «SPARQL vs. SQL - Intro,» [Online]. Available: http://www.cambridgesemantics.com/semantic-university/sparql-vs-sql-intro. [Consultato il giorno 2013].

[26] B. Adida, I. Herman, M. Sporny e M. Birbeck, «RDFa 1.1 Primer,» W3C, 2012. [Online]. Available: http://www.w3.org/TR/xhtml-rdfa-primer/.

[27] Apache Any23, «Architectural Overview,» 2013. [Online]. Available: http://any23.apache.org/developers.html.

[28] N. Freed e N. Borenstein, «RFC: 2046 - Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types,» Network Working Group, Novembre 1996. [Online]. Available: https://tools.ietf.org/html/rfc2046.

[29] W3C, «Document Object Model (DOM),» 2013. [Online]. Available: http://www.w3.org/DOM/.

[30] E. Orri, «Implementing a SPARQL compliant RDF Triple Store using a SQL-ORDBMS,» [Online]. Available: http://virtuoso.openlinksw.com/whitepapers/SPARQL%20RDF%20Store%20using%20SQL-ORDBMS.html. [Consultato il giorno 2013].

[31] The Apache Software Foundation, «ARQ - Application API,» 2013. [Online]. Available: http://jena.apache.org/documentation/query/app_api.html.

[32] Quantum GIS, «QGIS Features,» 2013. [Online]. Available: http://qgis.org/about-qgis/features.html.

[33] Python Software Foundation, «Python Programming Language,» 2013. [Online]. Available: http://www.python.org/.

[34] Digia Oyj, «Qt,» 2013. [Online]. Available: http://qt.digia.com/.

[35] Riverbank Computing Limited, «What is PyQt?,» 2013. [Online]. Available: http://www.riverbankcomputing.co.uk/software/pyqt/intro.

[36] Google, «Google Refine,» 2013. [Online]. Available: http://code.google.com/p/google-refine/.

[37] Servizio Sanitario Regionale Emilia-Romagna, «SaluteBologna,» 2013. [Online]. Available: http://www.salute.bologna.it/index.php?p=elencoFarmacie.

[38] Google, «GRELFunctions - List of functions supported by the Google Refine Expression Language (GREL),» 2013. [Online]. Available: http://code.google.com/p/google-refine/wiki/GRELFunctions.

[39] «DBpedia,» [Online]. Available: http://dbpedia.org/About.

[40] Linked Open Data Italia – LKDI ONLUS, «Linked Open Data Italia: Scuole Italiane,» [Online]. Available: http://www.linkedopendata.it/datasets/scuole.

[41] QuamtumGIS, «QGIS API Documentation,» 2013. [Online]. Available: http://www.qgis.org/api/.

[42] Comune di Firenze, «OpenData Firenze,» 2013. [Online]. Available: http://opendata.comune.fi.it/.

[43] Roma Capitale, «Open Data | Roma Capitale,» [Online]. Available: http://dati.comune.roma.it/.