UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

DIPARTIMENTO DI SCIENZE
FISICHE, INFORMATICHE, MATEMATICHE

Tesi di Laurea in

INFORMATICA

Progetto e Sviluppo di un'Applicazione di Question Answering su Knowledge Base eterogenee

Relatore
Prof. Riccardo Martoglia

Candidato
Sonia Moretti

Anno Accademico 2012/2013

A Marina e Vincenzo, per tutto.

Ringrazio il Relatore Prof. Riccardo Martoglia, per la disponibilità.
Un ringraziamento speciale a Mitch, per la pazienza.

Introduzione

La tesi di ricerca che ho affrontato riguarda lo studio, la progettazione e la realizzazione di un'applicazione di Question Answering volta ad interrogare contemporaneamente diverse basi di conoscenza.

Nello specifico è possibile sottoporre all'applicazione realizzata domande in linguaggio naturale che verranno utilizzate poi per la ricerca nelle Knowledge-Base per fornire come risultato una serie di pagine Web da poter visitare, contenenti la risposta alla domanda effettuatua.

Questo programma crea e utilizza basi di conoscenza realizzate appositamente per offrire all'utente la possibilità di navigare siti internet differenti senza dover cercare manualmente l'argomento di interesse su ognuno di essi; vengono già aperte le pagine inerenti alla domanda posta, utilizzando solo siti che offrono informazioni sicure ed aggiornate.

La scelta di affrontare questa argomentazione è nata dalla lettura di un articolo[1] che parla della difficoltà della sottomissione di query in linguaggio naturale ai siti internet o ai databases online, e accenna a un sito knowledge-base (*Wolframaplha*) che cerca di rispondere comunque a query in linguaggio naturale. Parla poi anche di altri siti che invece, nonostante manchino di questa sofisticata capacità, si appoggiano su un vastissimo database di informazioni che li rende quindi molto più 'ricchi' e che sono in grado di offrire all'utente delle risposte migliori, perchè più ricche di informazioni.

Il progetto che ho voluto realizzare ha come obiettivo primario quello di unire i pregi e minimizzare i difetti di queste due diverse tipologie di siti internet. Il suo scopo è infatti quello di sfruttare il question answering per rispondere automaticamente alle query in linguaggio naturale che l'utente sottoporrà, e di ottenere anche le migliori risposte, riguardanti l'argomento desiderato, interrogando le basi di conoscenza(knowledge-base) realizzate basandomi sui dataset offertemi da diverse sorgenti informative. Questi ultimi, infatti, contengono miliardi di triple RDF su entità e relazioni, ma non implementano nessun meccanismo di Question Answering, e non possono quindi recuperare gli elementi necessari per rispondere alle domande in linguaggio naturale e non posseggono le capacità per comporre risposte adeguate.

[1] '*IQ: The Case for Iterative Querying for Knowledge*' `http://www.cidrdb.org/cidr2011/Papers/CIDR11_Paper5.pdf`.

Se da un lato *Wolframalpha* implementa il Question Answering, dall'altro non possiede una vasta ed eterogenea base di informazioni su cui fare appiglio per offrire un'esauriente risposta alle query sottomesse. Gli altri siti Web, al contrario, offrono ampie basi di conoscenza ma non sanno gestire le query in linguaggio naturale. Interlacciandoli tra loro ho la possibilità di ottenere risposta alla sottomissione di query complesse e anche di utilizzare questo risultato come chiave di ricerca per estrapolare informazioni riguardanti tale argomento dalle basi di conoscenza degli altri siti che non accettano tale tipologia di query.

Obiettivi della Tesi

Gli obiettivi principali che mi prefisso di realizzare in questa mia tesi sono:

- **Recupero dei dataset** di ogni sito Web e la loro trasformazione in basi di conoscenza.
- **Studio, progettazione e realizzazione** di un algoritmo per l'interrogazione delle basi di conoscenza create partendo da una domanda in input in linguaggio naturale.
- **Studio, progettazione e realizzazione** di un algoritmo per la selezione di risposte ugualmente probabili risultanti dalla precedente interrogazione.
- **Progettazione e realizzazione** di un'interfaccia grafica user-friendly.

Ogni obiettivo verrà ampliamente approfondito nei capitoli di questa Tesi e verranno motivate le scelte implementative che sono state fatte durante la realizzazione del programma.

Struttura della Tesi

La tesi si suddividerà in diversi capitoli che sono ordinati per ordine cronologico, ossia seguiranno l'effettivo sviluppo del progetto fase per fase, come effettivamente è stato realizzato.

Capitolo 1 - Background. Introduzione ai formati di dati che verranno utilizzati nel corso della tesi; analisi e confronto delle sorgenti informative utilizzate nella stessa.

Capitolo 2 - Progetto del software. Analisi dei requisiti e progettazione iniziale delle funzionalità che si vogliono implementare per soddisfare gli stessi; diagrammi esemplificativi di ciò che si andrà a realizzare; progettazione dei database.

Capitolo 3 - Implementazione. Problematiche centrali incontrate durante lo sviluppo del programma; scelte implementative, metodi e tecniche utilizzate nella scrittura del codice, nel dettaglio: realizzazione dei database, traduzione delle domande inserite dall'utente, question answering, visualizzazione delle risposte migliori e realizzazione dell'interfaccia grafica.

Capitolo 4 - Analisi dei Risultati. Presentazione riassuntiva dei risultati ottenuti dall'applicazione.

Capitolo 5 - Conclusioni. Riassunto di cosa si è ottenuto e di cosa invece potrebbe essere migliorato.

Appendice A - Archivio dei codici. Archivio contenente la totalità del codice scritto.

Indice

Capitolo 1

Background

In questo capitolo verranno presentate tutte le informazioni che è necessario conoscere per capire il programma da me realizzato e che sarà oggetto di studio in questa tesi di laurea.

Verrà inizialmente presentato il modello RDF (Resource Description Framework) che viene utilizzato per rappresentare i metadati con cui lavorerò nel mio programma. Verranno poi descritte le scelte che sono state prese prima dell'inizio della creazione del programma vero e propio rigurdanti i siti internet da utilizzare con le relative caratteristiche che hanno contribuito alla loro scelta.

1.1 Il modello RDF

Il modello RDF (Resource Description Framework)[1] è uno strumento proposto da W3C (World Wide Web Consortium)[2] per organizzare, descrivere e interscambiare i metadati[3] relativi a una risorsa in un formato standard per consentirne il riutilizzo tra applicazioni che condividono le informazioni sul Web, senza che ci sia perdita di significato.

Lo scopo di RDF è, quindi, quello di generare informazioni che non siano destinate solo alla lettura, ma anche al riutilizzo da parte di altre applicazioni.

Le informazioni descritte attraverso RDF riguardano degli oggetti della realtà che è possibile identificare sulla rete attraverso un URI (Uniform Resource Identifier)[4], ossia attraverso un indirizzo Web univoco. L'oggetto scelto viene poi descritto aggiungendogli informazioni che lo descrivono, come ad esempio degli attributi o delle relazioni con altre entità.

Inoltre, il modello RDF è 'astratto', nel senso che non è legato ad una sola modalità di implementazione, quindi è possibile rappresentarlo attraverso diverse forme, come XML, OWL[5], etc. .

RDF è formato da due componenti:

- *RDF Model and Syntax*: descrive la struttura da utilizzare per descrivere le risorse e definisce con quale tipologia farlo, ossia quale formato di rappresentazione utilizzare (es. XML, OWL, etc.);
- *RDF Schema*: espone la sintassi per definire gli schemi e i vocaboli dei metadati.

Il data model RDF si basa su una *tripla* formata da tre oggetti, che sono i concetti fondamentali di questo standard:

- *Resource* (risorsa): ciò che viene descritto dall'RDF, può essere una risorda Web (immagini, file, pagine HTML, etc.) o anche esterna, ossia elementi che non appartengono direttamente al Web ma che si possono rintracciare tramite esso (un libro, un quadro, una persona). Identifica qualsiasi elemento che abbia un URI;
- *Property* (proprietà): indica una proprietà della risorsa, ossia una sua caratteristica o attributo ma può anche essere una relazione che lega la stessa ad altre risorse;

[1] Struttura logica per la descrizione di risorse.

[2] [...]organizzazione non governativa internazionale che ha come scopo quello di sviluppare tutte le potenzialità del World Wide Web. (Wikipedia)

[3] Un metadato è un'informazione utilizzata per descrivere più dati.

[4] [...]è una stringa che identifica univocamente una risorsa generica che può essere un indirizzo Web, un documento, un'immagine, un file, un servizio, un indirizzo di posta elettronica, ecc. L'URL è un URI, o più comunemente chiamato indirizzo web. (Wikipedia)

[5] Ontology Web Language

- *Value* (valore): è l'elemento che descrive la risorsa, indica cioè il valore della proprietà e descrive le caratteristiche della risorsa.

Per rappresentare un'informazione RDF si utilizza uno *statement* (asserzione), costituito dall'insieme di una risorsa, la sua proprietà e i valori per tale proprietà. Lo *statement* di una risorsa è quindi una *tripla* formata da un *soggetto* (resource), un *predicato* (property), e un *oggetto* (value).

In forma grafica lo *statement* viene rapprensentato tramite un grafo orientato, vedi **Figura 1.1**, in esso vediamo come le propietà sono rappresentate. L'arco collega il soggetto al predicato, rappresentati rispettivamente da un ellisse e da un rettangolo.

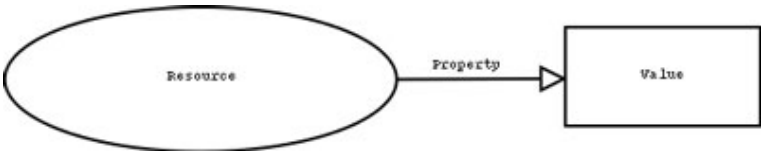

Figura 1.1: Statement RDF

1.1.1 Esempi RDF

Ipotizziamo di voler dire che un dato sito Web sia stato creato da una certa persona. Lo statement che creeremo descriverà quindi la caratteristica 'autore' di tale sito. Con RDF il concetto verrà così espresso:

- soggetto: `http:www.myhost.org/~mrossi`
- predicato: autore
- oggetto: Mario Rossi

Così facendo posso esprimere chiaramente il fatto che Mario Rossi è l'autore del suddetto sito, e il tutto verrà rappresentato graficamente come in **Figura 1.2**.

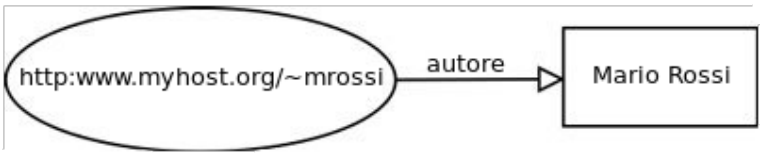

Figura 1.2: esempio statement RDF

RDF non pone limiti all'espansione dei grafi, si possono creare relazioni molto complesse inserendo nuovi predicati oppure considerando l'oggetto come un nuovo soggetto

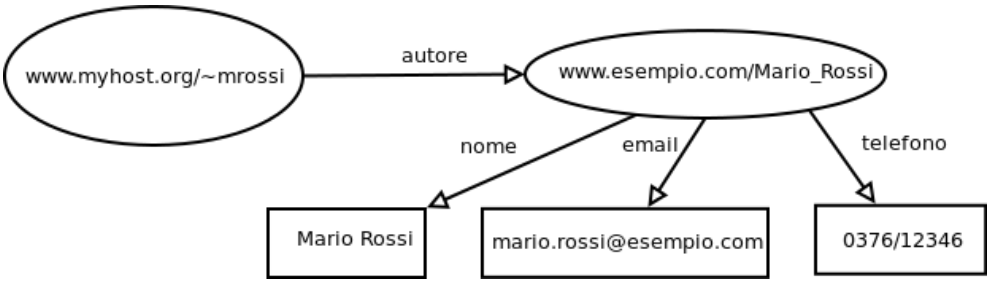

Figura 1.3: esempio statement RDF ampliato

al quale associare nuove proprietà. Per esempio, il grafo in **Figura 1.2** potrebbe venire espanso come in **Figura 1.3**.

In XML la sintassi estesa dell'esempio in figura **Figura 1.2** risulta così:.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <rdf:RDF
3   xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
4   xmlns:a="http://esempio.com/schema_autore">
5  <rdf:Description about="http://www.myhost.org/~mrossi">
6    <a:autore>
7      Mario Rossi
8    </a:autore>
9  </rdf:Description>
10 </rdf:RDF>
```

La descrizione del metadato si trova all'interno dell'elemento *<rdf:Description>*; l'attributo *about* identifica la risorsa vera e propria alla quale si riferisce il metadato, ossia l'URL `http://www.myhost.org/~mrossi`. La proprietà è invece descritta con il tag *<a:autore>* e il valore di quest'ultima è **Mario Rossi**.

Nella definizione dell'elemento *<rdf:RDF>* troviamo invece due namespace, quello relativo a RDF, alla riga 3, e quello relativo allo schema utilizzato per descrivere lo schema RDF, alla riga 4, ossia la semantica e le convenzioni per la regolazione delle proprietà presenti nella statement.

Se volessimo invece rappresentare in XML il secondo grafo, quello presente in **Figura 1.3**, lo faremmo nel seguente modo:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
  xmlns:a="http://esempio.com/schema_autore">
 <rdf:Description about="http://www.myhost.org/~mrossi">
   <a:autore
  rdf:resource="http://esempio.com/Mario_Rossi">
 </rdf:Description>
```

```
9
10   <rdf:Description about="http://esempio.com/Mario_Rossi">
11     <a:nome>Mario Rossi</a:nome>
12     <a:email>mario.rossi@esempio.com</a:email>
13     <a:telefono>037612346</a:telefono>
14   </rdf:Description>
15 </rdf:RDF>
```

In questo esempio vengono descritte due risorse: il sito internet e il suo autore. L'autore, descritto alla riga 10, ha tre proprietà: nome, email e telefono.

Le due risorse sono collegate tra loro mediante la relazione *rdf:resource* presente nella proprietà *<a:autore>*, che fa sì che la descrizione della seconda risorsa (l'autore), sia collegata alla prima, ossia a quella del sito Web.

Come ultimo esempio, voglio mostrare un dataset che utilizzo nel mio programma e che contiene i dati presenti sui siti Web di **DBpedia** e di **Freebase**, in formato RDF. Qui di seguito mostro, tramite un elenco, come vengono rappresentate le risorse nel file contenente il dataset di **Freebase**:

- `http://dbpedia.org/resource/Name`, rappresenta la risorsa descritta, ossia l'URL della risorsa stessa, un indirizzo Web del sito *DBpedia*;
- `http://www.w3.org/2002/07/owl#sameAs`, la relazione che lega la risorsa descritta con la sua proprietà, in questo caso una relazione di *sameAs*,
  ossia 'lo stesso di ' ;
- `http://rdf.freebase.com/ns/m.01006r`, il valore della proprietà, ossia l'URL per accedere ad una pagina di *Freebase* contenente la stessa risorsa descritta dalla prima URL, quella di *DBpedia*.

1.2 Le sorgenti informative

Partendo dall'articolo *'IQ: The Case for Iterative Querying for Knowledge'*[6], ho deciso di selezionare solo alcuni tra i siti presentati allo scopo di utilizzarne i databases[7] o, dove possibile, per interrogarli direttamente online.

La scelta è stata guidata da diverse motivazioni che verrano presentate dettagliatamente nei sottocapitoli che seguiranno.

I siti in questione sono:

- WolframAlpha[8]
- DBpedia[9]
- Freebase[10]
- CIA - the World Factbook[11]

La tabella in **Figura 1.4** mostra le differenze e le somiglianze tra le varie sorgenti da me selezionate.

[6] `http://www.cidrdb.org/cidr2011/Papers/CIDR11_Paper5.pdf`

[7] *database o base di dati:* indica un archivio dati, o un insieme di archivi, in cui le informazioni in esso contenute sono strutturate e collegate tra loro secondo un particolare modello logico.[...] (Wikipedia)

[8] `http://www.wolframalpha.com`

[9] `http://dbpedia.org/About`

[10] `http://www.freebase.com`

[11] `https://www.cia.gov/library/publications/the-world-factbook`

	CIA	FREEBASE	DBPEDIA	WOLFRAMPALPHA
A COSA RISPONDE	Informazioni sugli stati del mondo	Informazioni rigurdanti principalmente personaggi famosi/libri/film.Non risponde ad espressioni matematiche	Informazioni presenti su Wikipedia	Espressioni matematiche. Informazioni puramente scientifiche, permette di effettuare confronti tra due concetti
COME SI SOTTOMETTE LA DOMANDA	Selezione da un elenco contenente i nomi degli stati	MID, machineID che corrisponde alla risorsa cercata	Inserimento del concetto cercato, non consente l'inserimento di interrogazioni ma solo di parole chiave	Interrogazioni in linguaggio naturale
LINGUA DI INTERROGAZIONE	Inglese	Inglese	119 lingue disponibili	Inglese
DATI SCARICABILI	Formato HTML, PDF.	Database completo nel formato RDF	datasets e ontologie scaricabili nel formato RDF	Nessun dato scaricabile. È possibile scaricare l'output di una query ma solo con la versione 'pro' a pagamento.
ESEMPIO DI URL	https://www.cia.gov/library/publications/the-world-factbook/geos/it.html	http://www.freebase.com/m/03rjj	http://dbpedia.org/page/Italy	http://www.wolframalpha.com/input/?i=italy&dataset=

Figura 1.4: Tabella di confronto tra le varie sorgenti

1.2.1 WolframAlpha

Figura 1.5: logo WolframAlpha

	WOLFRAMPALPHA
A COSA RISPONDE	Espressioni matematiche. Informazioni puramente scientifiche, permette di effettuare confronti tra due concetti
COME SI SOTTOMETTE LA DOMANDA	Interrogazioni in linguaggio naturale
LINGUA DI INTERROGAZIONE	Inglese
DATI SCARICABILI	Nessun dato scaricabile. È possibile scaricare l'output di una query ma solo con la versione 'pro' a pagamento.
ESEMPIO DI URL	http://www.wolframalpha.com/input/?i=italy&dataset=

Figura 1.6: estratto delle caratteristiche

WolframAlpha è un sito internet che offre la possibilità agli utenti di interrogare lo stesso con query[12] in linguaggio naturale[13] e di ottenere una risposta precisa, nel caso ci sia, oltre ad una serie di informazioni di contorno per espandere il concetto.

[12][...]interrogazione da parte di un utente di un database [...] per compiere determinate operazioni sui dati. (Wikipedia)

[13]query espressa in un idioma, ad esempio l'inglese o l'italiano.

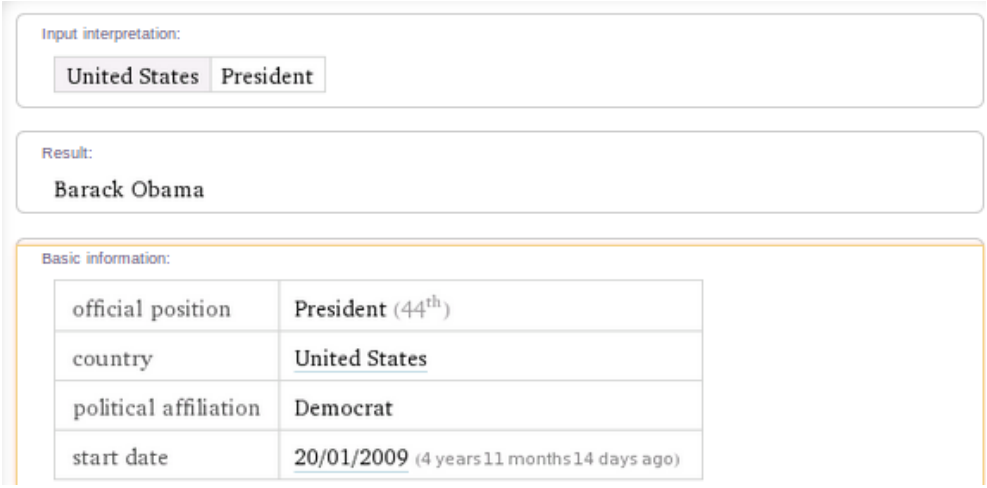

Figura 1.7: risposta alla domanda *'Who is the president of USA?'*

Il sito è in grado di rispondere ad una vasta gamma di domande appartenenti a diverse categorie, come politica, storia, astrologia, matematica, etc., offrendo una risposta ricca di informazioni di contorno ritenute importanti.

WolframAlpha per ampliare i suoi domini di conoscenza, oltre ad essere sempre aggiornato e ad appoggiarsi a fonti esterne per espandere le sue competenze, è tutt'ora in fase di sviluppo strutturale, ad esempio, è appena stata introdotta la possibilità di avere una risposta a domande che hanno insito un grado di parentela tra persone (es. *'Who is the father of Miley Cyrus?'*).

Un altro aspetto tecnico è la possibilità di scaricare le pagine di WolframAlpha in formato HTML[14] e quindi di poter accedere ai risultati che offre anche tramite un'applicazione terza che è in grado di elaborare questo formato di file.

Il programma da me realizzato utilizzerà WolframAlpha principalmente per la possibilità di poter inserire domande in linguaggio naturale e di poterne ricavare e riutilizzare le risposte che offre, in altre parole per implementare il Question Answering.

[14]HyperText Markup Language

1.2.2 DBpedia

Figura 1.8: logo DBpedia

	DBPEDIA
A COSA RISPONDE	Informazioni presenti su Wikipedia
COME SI SOTTOMETTE LA DOMANDA	Inserimento del concetto cercato, non consente l'inserimento di interrogazioni ma solo di parole chiave
LINGUA DI INTERROGAZIONE	119 lingue disponibili
DATI SCARICABILI	datasets e ontologie scaricabili nel formato RDF
ESEMPIO DI URL	http://dbpedia.org/page/Italy

Figura 1.9: estratto delle caratteristiche

'DBpedia is a crowd-sourced community effort to extract structured information from Wikipedia and make this information available on the Web.'[15]

DBpedia offre la possibilità di scaricare la propria ontologia[16] e il proprio dataset[17]in 119 lingue diverse.

Esiste anche, oltre alla versione inglese del suddetto sito, una versione italiana: *'La DBpedia Italiana è un progetto aperto e collaborativo per l'estrazione e il riutilizzo di*

[15]cit. `dbpedia.org/About`

[16][...]rappresentazione formale, condivisa ed esplicita di una concettualizzazione di un dominio di interesse. (Wikipedia)

[17]collezione di dati.

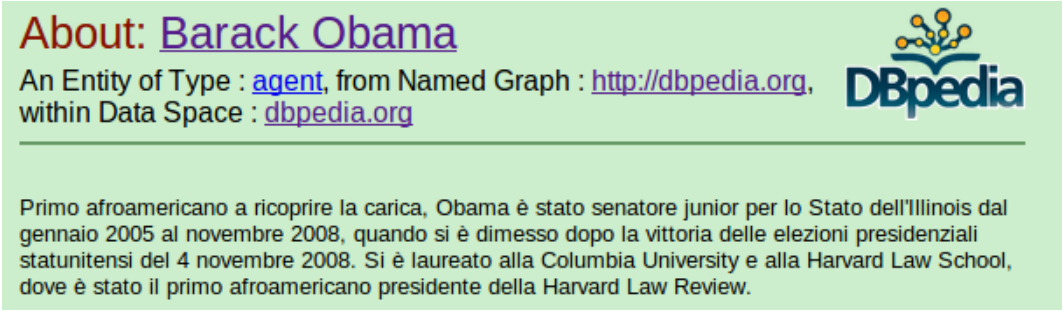

Figura 1.10: esempio di interrogazione di DBpedia

informazioni semanticamente strutturate dalla versione italiana di Wikipedia.'[18] che però ho deciso di non utilizzare nel mio progetto. La scelta di utilizzare la DBpedia in lingua inglese è stata dettata dal fatto che tutti gli altri siti che utilizzo sono anch'essi in lingua inglese e non offrono quindi una versione italiana o la possibilità di scaricare i loro dati in italiano. Anche WolframAlpha non è in grado di rispondere ad interrogazioni in una lingua diversa da quella inglese.

Nel mio programma ho deciso di poter far inserire all'utente le interrogazioni in qualsiasi lingua si voglia per rendere più precisa l'interrogazione; sulle modalità e le motivazioni di questa scelta ne parleremo nel **Capitolo 2 Progetto del software**. I risultati però saranno mostrati solo su pagine web in lingua inglese. Questo perchè, se avessi deciso di utilizzare la DBpedia italiana non avrei potuto sincronizzare al meglio i risultati di questo database con quelli degli altri che sarebbero stati comunque in inglese (ad esempio, il problema dei doppi significati che alcune parole assumono in una lingua e non nell'altra). Inoltre, per una ragione non meno forte, ho scelto di usare l'inglese per la prospettiva di un utilizzo futuro della mia applicazione, in quanto l'inglese è comunemente riconosciuto come lingua franca.

Il grande pregio di DBpedia è quello di offrire vastissimi datasets scaricabili. Oltre a quelli contenenti i dati inerenti alle sue stesse pagine web, ve ne sono molti contenenti i links a pagine Web di siti internet differenti. Questi datasets contengono informazioni aggiuntive sugli argomenti già presenti su Dbpedia.

Ho, quindi, deciso di utilizzare DBpedia come fonte primaria nel mio progetto, in quanto contiene il maggior e più completo dataset. Da questo sito ho deciso anche di scaricare il dataset dei collegamenti a Freebase[19] per una ragione di comodità (vedasi il capitolo seguente).

[18] cit. `it.dbpedia.org`

[19] `http://wiki.dbpedia.org/Downloads39#links-to-freebase`

1.2.3 Freebase

Figura 1.11: logo Freebase

	FREEBASE
A COSA RISPONDE	Informazioni rigurdanti principalmente personaggi famosi/libri/film.Non risponde ad espressioni matematiche
COME SI SOTTOMETTE LA DOMANDA	MID, machineID che corrisponde alla risorsa cercata
LINGUA DI INTERROGAZIONE	Inglese
DATI SCARICABILI	Database completo nel formato RDF
ESEMPIO DI URL	http://www.freebase.com/m/03rjj

Figura 1.12: estratto delle caratteristiche

Freebase offre *'A community-curated database of well-known people, places, and things'*[20]. Nel complesso Freebase offre però un dataset meno ampio di quello di DBpedia, pertanto ho deciso di metterlo in seconda posizione nell'ordine dei database da interrogare.

Freebase da anch'esso la possibilità di scaricarne i dati ma ho deciso di non sfruttarla e di utilizzare i links dal dataset di DBpedia. Questa scelta è stata dettata da una motivazione di comodità, in quanto gli URL[21] di DBpedia contengono un chiaro riferimento

[20] cit. `www.freebase.com`
[21] uniform resource locator

all'entità cercata (es. `dbpedia.org/page/Barack_Obama`), mentre quelli di Freebase contengono un MID[22], che è quindi indecifrabile (es.`www.freebase.com/m/02mjmr`).

Uno stratagemma, che spiegherò più concretamente nel **Capitolo 3 Implementazione**, che ho adottato per avere un'intelleggibilità degli URL di Freebase è stato quello di scaricarne il dataset da Dbpedia, il che mi ha consentito di avere una lista di tutti i collegamenti tra le risorse di Dbpedia stesso alle entità presenti in Freebase. In questo modo ho ottenuto sempre gli URL contenenti i MID ma però affiancati dagli URL più leggibili di Dbpedia.

Figura 1.13: esempio di interrogazione di Freebase.

1.2.4 CIA - the World Factbook

Figura 1.14: logo CIA

'The World Factbook provides information on the history, people, government, economy, geography, communications, transportation, military, and transnational issues for 267 world entities.'[23].

[22] *A mid or machine id is a short form of id for any Freebase topic.* (`wiki.freebase.com/wiki/Machine_ID`

[23] cit. `www.cia.gov/library/publications/the-world-factbook`

	CIA
A COSA RISPONDE	Informazioni sugli stati del mondo
COME SI SOTTOMETTE LA DOMANDA	Selezione da un elenco contenente i nomi degli stati
LINGUA DI INTERROGAZIONE	Inglese
DATI SCARICABILI	Formato HTML, PDF.
ESEMPIO DI URL	https://www.cia.gov/library/publications/the-world-factbook/geos/it.html

Figura 1.15: estratto delle caratteristiche

Il sito CIA è proprio il sito ufficiale della Central Intelligence Agency degli Stati Uniti d'America ed è pertanto il più attendibile ed aggiornato per quanto riguarda le informazioni geo- politiche di tutti gli stati del mondo. Per questo motivo ho deciso di utilizzarlo nel mio programma e di mostrarlo come primo risultato nel caso vengano fatte interrogazioni in cui ci sia una occorenza riferibile al nome o al dominio di primo livello di uno stato del mondo.

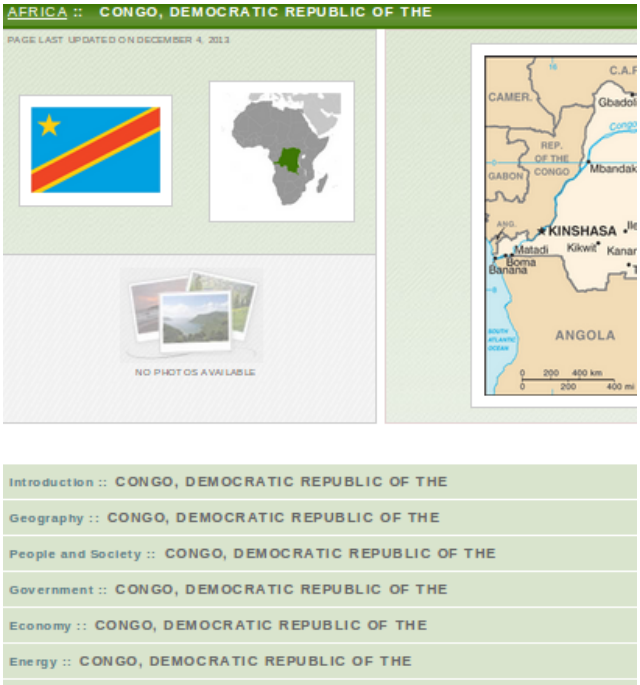

Figura 1.16: esempio di interrogazione di CIA.

Capitolo 2

Progetto del software

Analizzati nel dettaglio, nel capitolo precedente, le sorgenti informative di cui si dispone, si vuole ora illustrare la fase di progettazione vera e propria. In questo capitolo verranno mostrate, seguendo un ordine temporale, le operazioni e le scelte che sono state prese per arrivare ad un programma di interrogazione di knowledge base. Sempre in questo capitolo non verrano illustrate le scelte implementative effettuate durante la realizzazione del mio programma, questa parte verrà dettagliatamente spiegata nel capitolo successivo, **3 Implementazione**.

2.1 Analisi dei requisiti

Il programma che voglio realizzare, che chiamerò *Answering*, data una domanda inserita dall'utente, deve saper offrire dei link a delle pagine Web, appertenenti a siti attendibili, nei quali trovare informazioni rigurdanti la domanda posta.

La domanda che l'utente dovrà inserire deve essere in linguaggio naturale, nella sua lingua, in questo modo sarà possibile formulare correttamente anche richieste complesse.

Il programma deve rispondere alla domanda inserita dall'utente(Question Answering) e offrire la possibilità di visitare tutte le pagine Web nelle quali ha trovato una risposta.

Ciò che *Answering* deve essere in grado di fare è quindi:

- consentire l'inserimento di una domanda espressa in una qualsivoglia lingua;
- trovare una risposta a tale domanda (Question Answering);
- utilizzare la risposta ottenuta come chiave di ricerca nelle basi di conoscenza;
- restituire tutti i risultati ottenuti, meglio se in ordine di rilevanza.

Nei capitoli che seguiranno, verrà analizzato il processo di progettazione dei requisiti qui sopra elencati, spiegando tutte le scelte che sono state prese con le relative motivazioni.

2.2 Progettazione DB

In primo luogo è stato necessario identificare le sorgenti informative da utilizzare. Esse sono descritte nel **Capitolo 1.2 Le sorgenti informative**, e comprendono i siti Web di *DBpedia*, *Freebase*, *Wolframalpha* e *CIA*.

Di quei siti dei quali è possibile scaricarne il dataset, ossia *DBpedia*, *Freebase* e *CIA*, si è subito proceduto alla creazione di un relativo database interrogabile dal programma stesso.

Di *Wolframalpha*, invece, non è possibile scaricarne il dataset ma è in compenso consentito all'utente di disambiguare il risultato ottenuto direttamente dalla sua pagina Web, e questo ci permette di non aver bisogno della creazione di un suo database per la ricerca di disambiguazioni.

Per quanto riguarda il sito della *CIA*, non ho scaricato un dataset ma ho preferito scaricare una pagina HTML dal sito stesso ed estrapolarne le informazioni per ricreare gli URL per accedere alle altre pagine del sito in questione. Questo perchè non essendo il numero degli URL possibili eccessivamente vasto, questa risulta un'operazione più veloce.

In vista del fatto che per ogni sito Web in questione possedevo un dataset, ho deciso di creare per ognuno di essi un database indipendente per consentire, in vista di un

possibile ampliamento futuro del programma, l'inserimento di altri database senza dover effettuare modifiche a quelli già esistenti.

Si passerà, nel capitolo seguente, a mostrare lo schema che è stato adottato per la creazione delle basi di conoscenza che utilizzerò nel mio programma.

2.2.1 Diagramma E/R

Ho modellato quindi ognuno di essi con il modello E/R[1]. Nelle figure sottostanti ho rappresentato le entità di ogni database con i relativi attributi.

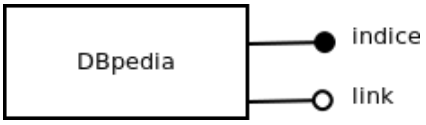

Figura 2.1: diagramma E-R del database di DBpedia

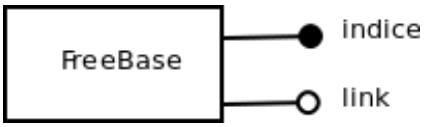

Figura 2.2: diagramma E-R del database di Freebase

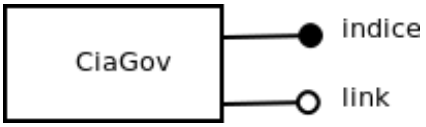

Figura 2.3: diagramma E-R del database di CIA

Per ogni entità rappresentata gli attributi sono: **indice**, che è anche la chiave[2], e **link**. Entrambe le proprietà sono obbligatorie, **indice** è anche unica, ossia tutte le istanze della classe assumono un valore diverso. **Indice** conterrà le parole chiave che identificano l'entità, ed è su queste che verrà effettuata la ricerca dei risultati, mentre **link** conterrà la parte finale dell'indirizzo Web della risorsa. È con il valore presente in **link** che otterremo l'URL completa per accedere alla risorsa identificata in **indice** sul suo relativo sito Web.

Come già accennato inizialmente, ho deciso di lasciare le entità slegate tra loro e di creare quindi, per ognuna di esse, un diverso database. Questa scelta, oltre che per motivi

[1]In informatica, nell'ambito della progettazione dei database, il modello entity-relationship[...] è un modello per la rappresentazione concettuale dei dati ad un alto livello di astrazione[...] (Wikipedia)

[2]la proprietà chiave identifica in modo univoco la singola istanza di entità

di ampliamenti futuri, è stata presa anche per facilitare l'interrogazione degli stessi da parte di *threads*[3] (per la motivazione della scelta dell'uso di threads si veda **Capitolo 3 Implementazione**). Infatti, nel mio programma, viene istanziato un thread per ogni database da interrogare, in modo da velocizzare il tempo di ricerca. Database separati, infatti, permettono l'utilizzo di threads concorrenti senza aver bisogno di sincronizzarli. Si evita di dover realizzare, quindi, dei sistemi di attesa degli stessi per impedire gli accessi concorrenti alla stessa risorsa.

2.3 Diagrammi delle attività

Il funzionamento del programma è stato modellato utilizzando il diagramma delle attività[4] qui di seguito illustrato (per maggior chiarezza sono state delimitate le operazioni riservate all'utente dalla linea verticale tratteggiata).

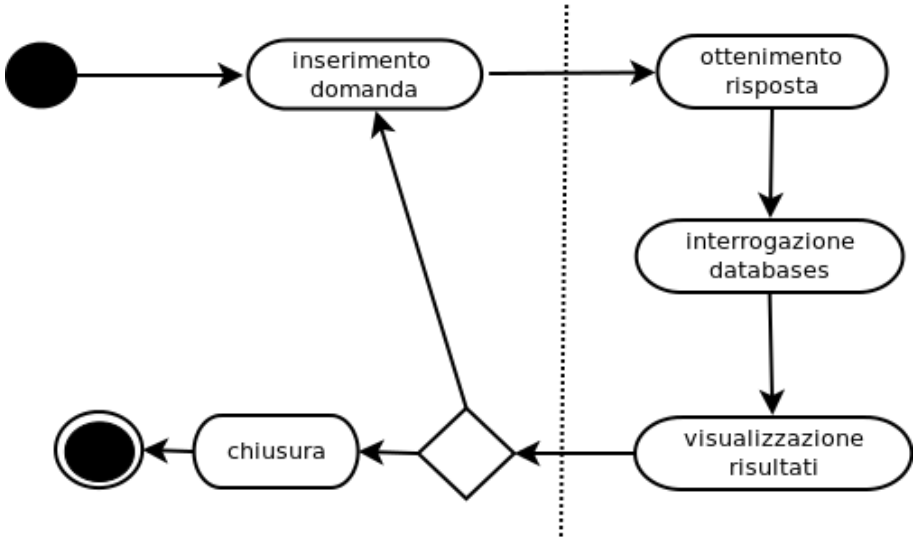

Figura 2.4: diagramma delle attività

Il diagramma modella i requisiti presentati nel **capitolo 2.1 Analisi dei requisiti** . L'utente inserisce la domanda in una qualsiasi lingua e questa attraversa un passaggio di Question Answering per giungere ad una risposta. In questo passaggio verrà utilizzato il sito *Wolframalpha*, che prende in input domande in linguaggio naturale e che restituisce

[3]Un thread o thread di esecuzione, in informatica, è una suddivisione di un processo in due o più filoni o sottoprocessi, che vengono eseguiti concorrentemente da un sistema di elaborazione monoprocessore (multithreading) o multiprocessore. (Wikipedia)

[4]L'Activity Diagram è un diagramma definito all'interno dello Unified Modeling Language (UML) che definisce le attività da svolgere per realizzare una data funzionalità. Può essere utilizzato durante la progettazione del software per dettagliare un determinato algoritmo. (Wikipedia)

una riposta che verrà utilizzata per effettuare ricerche sui database e, infine, i risultati ottenuti da questa fase verranno mostrati all'utente.

Il problema che ci troviamo ora ad affrontare è quello della lingua: la domanda inserita dall'utente è in una lingua, di default è impostato l'italiano, mentre il sito *Wolframalpha*, è interamente in lingua inglese. Anche le basi di conoscenza con cui dobbiamo lavorare hanno tutte le loro entità espresse in lingua inglese. Quello che occorre è quindi una fase di **traduzione**, che trasformi la frase inserita in input nel suo corrispettivo in lingua inglese.

La domanda viene fatta quindi passare attraverso una fase di traduzione, svolta dal programma stesso e alla quale l'utente non può partecipare; come si può vedere infatti dalla stessa figura, le azioni che può intrapprendere l'utente sono delimitate dalla linea verticale tratteggiata, e la traduzione non è tra queste.

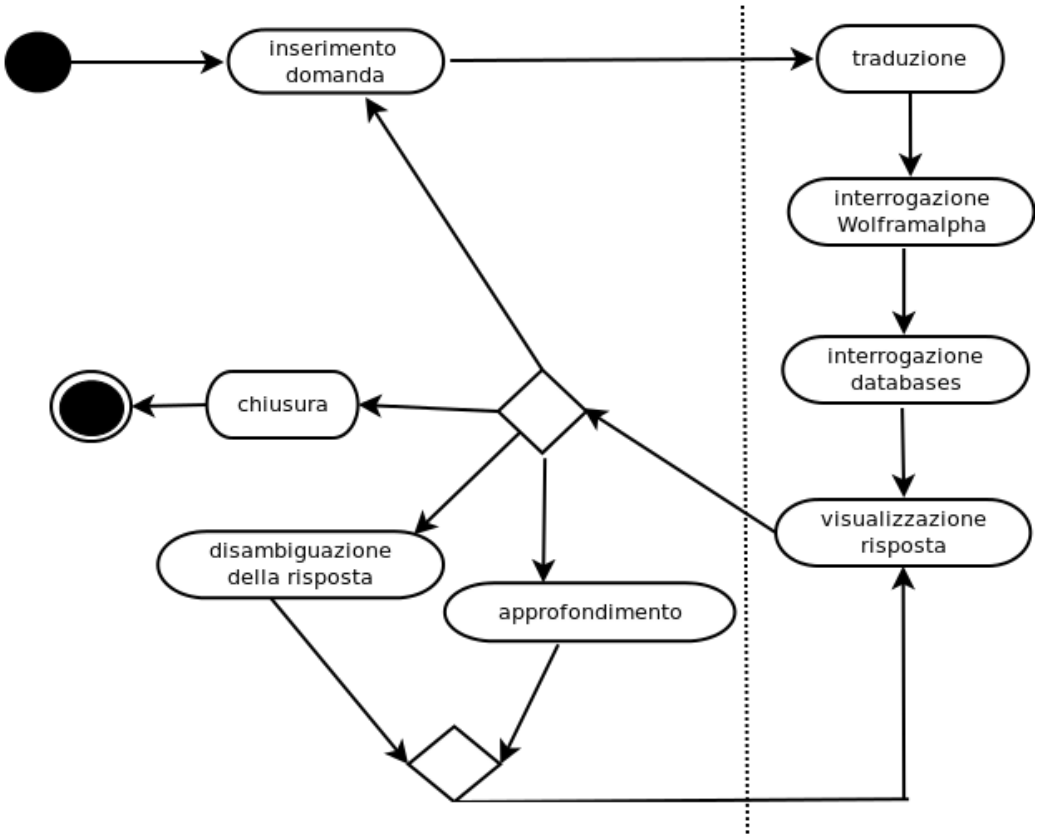

Figura 2.5: diagramma delle attività

Solo in seguito alla fase di traduzione si passa poi all'interrogazione del sito *Wolframalpha* per cercare di ottenere una risposta alla domanda fatta e si utilizza quest'ultima per interrogare le Knowledge Base degli altri siti Web.

La risposta viene visualizzata tramite l'apertura di una pagina Web, la prima che verrà mostrata sarà quella risultante dall'interrogazione di *Wolframalpha*.

Da questa pagina sarà poi possibile disambiguare direttamente il risultato ottenuto, se vi sono doppi significati per la domanda sottoposta, scegliendo da una serie di opzioni che *Wolframalpha* offre all'utente nella sua interfaccia.

Ora sta all'utente stesso dover decidere quale azione intrapprendere tra quelle disponibili. A parte la chiusura del programma e l'inserimento di una nuova domanda, che farebbe ripartire il ciclo appena descritto ex novo, l'utente può decidere di approfondire la risposta ottenuta o di disambiguarla scegliendo da un elenco di possibilità. In entrambi i casi, comunque, si procederà all'apetura di una nuova pagina Web.

Questa scelta è stata presa in quanto, nella maggior parte dei casi, ad una parola corrispondono più significati. L'esempio più significativo è quello della parola inglese '*mouse*': se si interroga *DBpedia* o *Freebase*, compare una pagina di disambiguazione del risultato, in quanto la parola può assumere diversi significati. Quando si seleziona la risposta più 'coerente' da mostrare subito all'utente, non è detto che contenga il significato che voleva esprimere l'utente stesso. Per questo motivo è stato aggiunto un tasto di disambiguazione del risultato, che mostra tutte le possibilità di disambiguazione del termine, all'interno del sito Web attuale.

Inoltre l'utente potrebbe voler scorrere tutti i siti contententi una risposta riguardante la domanda fatta, per questo motivo è stata aggiunta la scelta di approfondimento, che apre una nuova pagina Web di un altro sito, e che aggiorna la lista delle disambiguazioni possibili per il sito in questione.

Entrando ancor più nel dettaglio, **Figura 2.6**, vediamo come la prima fase diventi il passaggio della domanda sottoposta dall'utente in italiano al sito **Google translate**, per ottenere la stessa tradotta in inglese.

La scelta di utilizzare Google Translate è stata presa in quanto è il miglior sito gratuito di traduzione online al quale è possibile sottoporre le interrogazioni direttamente tramite l'URL e scaricarne la pagina HTML da cui estrapolarne la domanda tradotta.

Successivamente avviene l'interrogazione del sito *Wolframalpha*, passandogli la domanda in inglese ed ottenendo, nella maggior parte dei casi, una risposta.

Lo scopo che ci poniamo ora è quello di ridurre ulteriormente la risposta o la domanda, nel caso la prima non sia disponibile, in una lista di parole chiave, eliminando quindi quelle inutili e riducendo le rimenenti alla loro forma base.

Questa soluzione è stata adottata principalmente per quelle occasioni in cui il sito *Wolframalpha* non è in grado di fornirci una risposta e si necessiti di apporre modifiche alla domanda sottomessa dall'utente.

A questo punto quello che ho è la lista di parole chiave estrapolate dalla risposta di *Wolframalpha* o dalla domanda dell'utente.

Il passaggio che segue è quello vero e proprio dell'interrogazione dei database che avviene in modo parallelo, grazie all'utilizzo di **threads** concorrenti.

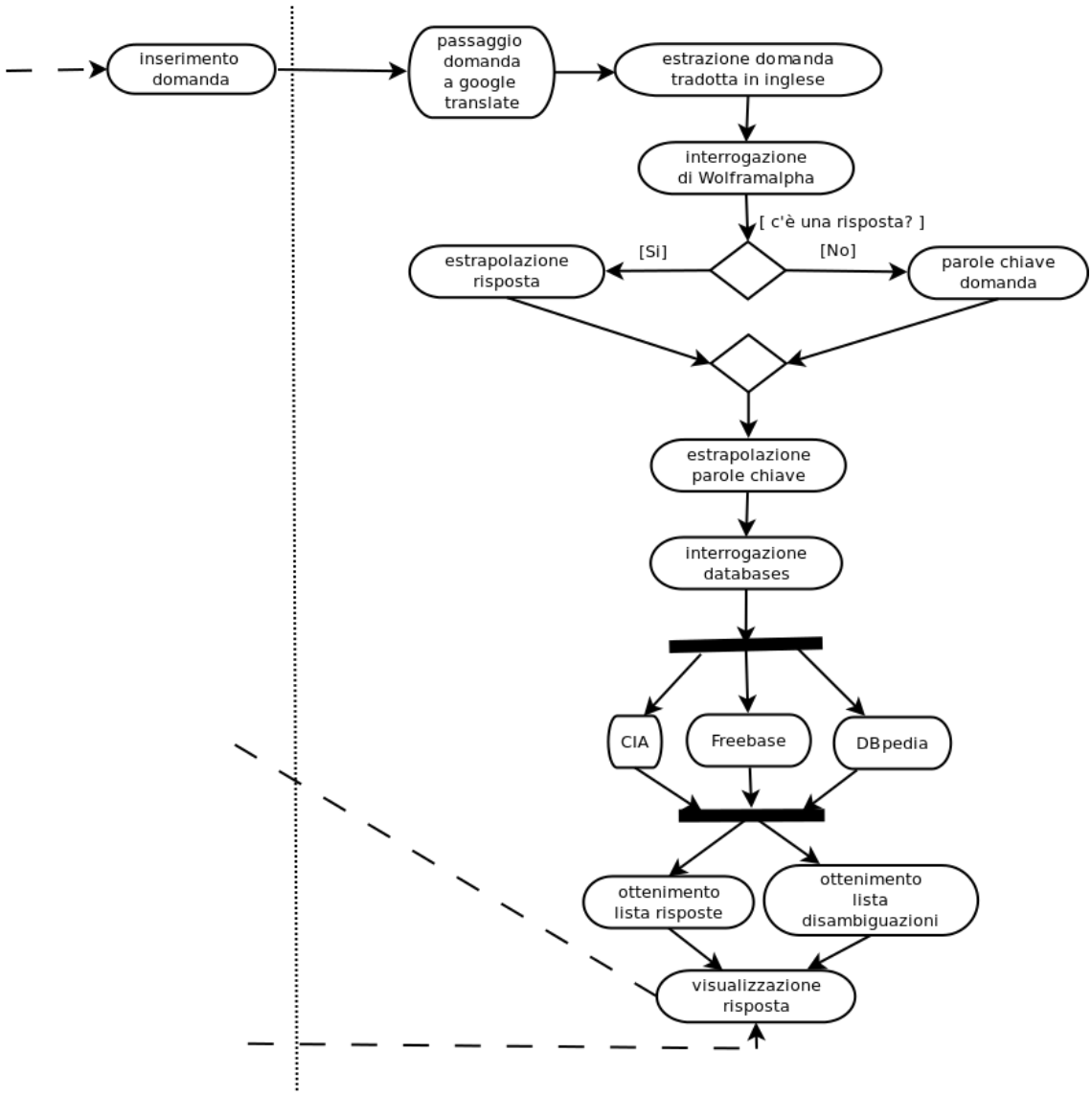

Figura 2.6: diagramma delle attività

Le basi di conoscenza su cui lavorare sono di notevoli dimensioni e i risultati che restituiscono sono spesso numerosi e necessitano di notevoli scremature per eliminare quelli che non sono di interesse riguardo la domanda fatta.

La scelta di lavorare con i threads è stata, quindi, dettata da un problema di tempistica: dividendo il lavoro dell'interrogazione dei database e della selezione dei risultati in parti uguali e concorrenti, i tempi di attesa per ottenere una risposta e far ritornare il controllo al programma principale, saranno ridotti notevolmente. Ognuno di questi threads, infatti, lavora in modo indipendente dagli altri e alla fine del suo lavoro, resti-

tuisce al programma principale il risultato da mostrare per il sito Web da lui analizzato e la lista delle sue disambiguazioni.

Alla fine ottengo, quindi, una lista contenente l'URL migliore di ogni sito Web nel quale è stata trovata una risposta alla domanda sottoposta, e una lista contenente le disambiguazioni possibili per le risposte ottenute da ogni sito Web.

Capitolo 3

Implementazione

Essendosi il lettore fatto, a questo punto della lettura, un'idea su ciò che verrà realizzato nel corso di questa tesi, si passa ora alla spiegazione dettagliata dell'implementazione del programma 'Answering'.

Verrà presentato e commentato il codice del programma, seguendo l'ordine temporale di utilizzo dello stesso: partendo dalla creazione dei database per passare poi alla loro interrogazione, fino ad arrivare alla visualizzazione dei risultati e alla possibilità di disambiguare gli stessi.

3.1 Il linguaggio di programmazione

Ho deciso di utilizzare Python come linguaggio per la scrittura del mio programma per diversi motivi. In primis, l'ho scelto in quanto consente una scrittura rapida e gestisce automaticamente i dettagli ostici riguardanti la gestione della memoria. Inoltre, offre molte librerie che consentono di utilizzare funzionalità in modo automatico, cioè senza doverne riscrivere il codice. Tra queste, quelle maggiormente utilizzate sono impiegate per interfacciarsi con i database e con le pagine Web. Infine, ho scelto Python perchè è un linguaggio molto portabile e conosciuto, il che faciliterà eventuali modifiche future anche da parte di altri.

3.2 Dai dataset alle basi di conoscenza

Una volta trovati i dataset più idonei per il mio programma ho dovuto trasformarli in database per poterli interrogare e questo ha comportato un'operazione di 'pulizia' dalle informazioni non necessarie.

Nei sottocapitoli che seguiranno voglio mostrare il codice relativo ai due passaggi fondamentali che sono stati implementati per creare i database a partire dai file di testo scaricati, contenenti i dataset o la pagina HTML.

3.2.1 I dataset

Alla prima apertura di *Answering*, viene chiesto all'utente di scaricare due file contententi i rispettivi dataset di *Freebase* e di *DBpedia*.

Essi sono necessari al funzionamento del programma e ad ogni apertura dello stesso, avviene un controllo per verificare che tali file siano presenti nella cartella di destinazione del programma. Senza di essi il programma non parte.

Inizialmente, i file che possediamo sono quindi *page_ids_en.ttl* per *DBpedia* e *freebase_link.nt* per *Freebase*, scaricabili rispettivamente agli indirizzi `http://downloads.dbpedia.org/3.8/en/page_ids_en.ttl.bz2` e `http://downloads.dbpedia.org/3.8/links/freebase_links.nt.bz2`.

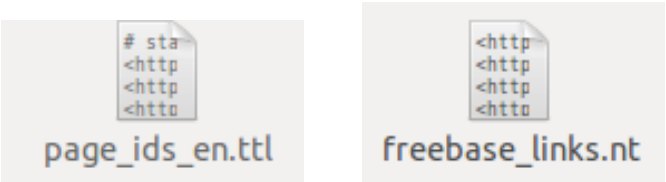

Figura 3.1: file iniziali contenenti i dataset

I file appena citati sono nel formato RDF (vedi **Capitolo 1.1 Il modello RDF**) e contengono quindi per ogni entità presente in *DBpedia* e in *Freebase* una riga nel corrispettivo file composta da tre URI distinte.

Per passare da questi file ai database ho prima creato due funzioni in Python che li 'snellissero' in automatico e creassero due nuovi file contenenti solo le informazioni necessarie a questo scopo. I file sono infatti di grandi dimensioni e richiedono non poco tempo per essere scanditi riga per riga. In questo modo, il tempo di elaborazione e le dimensioni dei file finali da utilizzare per creare i database, si riducono di molto.

```
1  def Dwiki(self):
2    Dwiki=open('DictWiki.txt','w')
3    with open('page_ids_en.ttl') as wiki:
4      for line in wiki:
5        if not line.startswith('#'):
6          parola=line.split(None,1)
7          parola=parola[0]
8          parola=parola.rstrip('>\n')
9          parola=parola.replace('<http://dbpedia.org/resource/','')
10         try:
11           b=unicode(parola)
12           Dwiki.write(b+'\n')
13         except:
14           pass
15   Dwiki.close()
```

```
1  def Dfree(self):
2    Dfree=open('DictFree.txt','w')
3    with open('freebase_links.nt') as free:
4      for line in free:
5        siti=line.split(None,3)
6        sito=siti[2]
7        sito=sito.lstrip('<http://rdf.freebase.com/')
8        sito=sito.rstrip('>')
9        sito=sito.replace('ns/','')
10       sito=sito.replace('.','/')
11       parola=siti[0].replace('<http://dbpedia.org/resource/','')
12       parola=parola.rstrip('>')
13       Dfree.write(parola+' '+sito+'\n')
14   Dfree.close()
```

Per ogni riga dei file si utilizza la funzione *line.split* per separare le URI presenti e selezionarne una o più. Nel caso di *DBpedia*, si memorizza solo la prima URI, contentene l'indirizzo della risorsa sul sito stesso di DBpedia, mentre, per *Freebase* si memorizza la prima URI (memorizzata nella variabile **parola**) ma anche l'ultima, ossia quella contenente l'indirizzo della risorsa sul sito Freebase (memorizzata nella variabile **sito**).

Alle variabili **parola** e **sito** vengono applicate delle modifiche per isolare solo l'ultima parte delle URI che contengono, quella che identifica precisamente l'entità cercata per

DBpedia (es. *Barack_Obama*), e il MID della risorsa per *Freebase*.

I file risultanti sono *DictWiki.txt* e *DictFree.txt*. Essi contengono gli **indici** e i **link** da utilizzare durante la creazione dei rispettivi database.

Per la creazione del database di *CIA* ho proceduto invece diversamente. Ho deciso di non scaricarne il dataset da *DBpedia* ma di crearlo a partire dalla pagina HTML del sito stesso. Questa scelta è stata presa in quanto la quantità di dati è molto ridotta, il sito Web, infatti, contiene informazioni riguardanti solamente gli Stati mondiali. Inoltre, all'apertura del programma **Answering** avviene sempre un controllo sulla presenza dei files di 'configurazione' iniziali, e quindi anche del file TXT di *CIA*. Nel caso questo manchi, viene nuovamente scaricata la pagina HTML dal sito Web e viene ricreato il database. Questa scelta offre, quindi, la possibilità di aggiornare la lista degli indirizzi Web nel caso ci siano dei cambiamenti importanti nel sito stesso senza dover aspettare che *DBpedia* aggiorni i suoi dataset, ma si potrà aggiornare il tutto semplicemente eliminando un file dalla cartella di destinazione del programma.

La pagina Web da scaricare per ottenerne il codice HTML è stata selezionata casualmente, in quanto in tutte le pagine del sito Web è presente un elenco degli Stati selezionabili e dei relativi domini di primo livello nazionali[1] da apporre all'indirizzo della pagina Web per crearne il *pathname*[2], che indirizza alla risorsa cercata (es. `https://www.cia.gov/library/publications/the-world-factbook/geos/be.html` per il Belgio).

```
1  def Dcia(self):
2    sock=urllib.urlopen('https://www.cia.gov/library/publications/the-
       world-factbook/geos/be.html')
3    htmlSource=sock.read()
4    sock.close()
5
6    stringa='<option value="">Please select a country to view</option>'
7    indice=htmlSource.find(stringa)
8    result=htmlSource[(indice+len(stringa)):]
9    indice=result.find('</select>')
10   result=result[:indice]
11
12   result=result.replace('<option value="../geos/','')
13   result=result.replace(' </option>','')
14   result=result.replace('\t','')
15   result=result.replace('\n\n','\n')
16   result=result.replace('.html">','')
17   result=result.lstrip('\n')
18
19   Dcia=open('DictCia.txt','w')
20   Dcia.write(result)
21   Dcia.close()
```

[1][...]usati da uno stato o una dipendenza territoriale. È costituito da due lettere, per esempio jp per il Giappone e it per l'Italia[...]. (Wikipedia)

[2]Percorso nel file system del server che identifica la risorsa.

Per lavorare con le pagine Web ho utilizzato il modulo **urllib**: *'Questo modulo fornisce un'interfaccia di alto livello per estrarre dati attraverso il World Wide Web. In particolare, la funzione urlopen() è simile alla funzione built-in open(), ma accetta Universal Resource Locators (URL) invece dei nomi di file.'*[3].

Una volta aperta la pagina Web con il medoto **urlopen**, ne memorizzo il codice HTML, per utilizzarlo successivamente per estrapolarne la lista di Stati e domini. Questo avviene nelle righe dalla 6 alla 17, in cui isolo l'inizio dell'elenco degli Stati e dei rispettivi domini, da usare per completare l'URL per accedere all'entità sul sito della *CIA*, e applico delle modifiche al risultato, per eliminare i caratteri simbolici. Ottenengo infine il file *DictCia.txt*, contenente il dataset *sigla del dominio - Stato* (**Figura 3.2**).

```
ac Antigua and Barbuda
xq Arctic Ocean
ar Argentina
am Armenia
aa Aruba
at Ashmore and Cartier Islands
zh Atlantic Ocean
as Australia
au Austria
aj Azerbaijan
bf Bahamas, The
```

Figura 3.2: parte del file *DictCia.txt*

Il passaggio successivo è quello dai file in **Figura 3.3**, ossia i file appena ottenuti grazie a questa prima fase di 'scrematura' dei risultati, a quelli in **Figura 3.4**, ossia ai loro rispettivi database.

Figura 3.3: files contenenti i dataset modificati

3.2.2 Realizzazione database

Allo scopo di creare i database dei siti Web, ho importato il modulo **sqlite3**: *'SQLite is a C library that provides a lightweight disk-based database that doesn't require a separate*

[3]cit. `docs.python.it/html/lib/module-urllib.html`

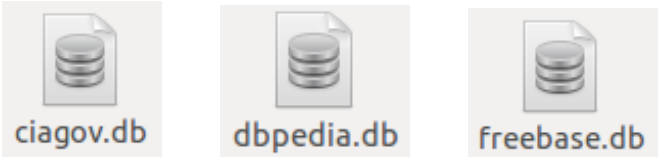

Figura 3.4: files contenenti i database

server process and allows accessing the database using a nonstandard variant of the SQL query language.'[4].
SQLite è, quindi, una libreria software che permette di implementare un DBMS SQL (Database Management System) [5] da poter incorporare all'interno di applicazioni.

Qui di seguito verranno mostrate le tre funzioni Python utilizzate per creare i tre database dei tre siti internet: *DBpedia*, *Freebase*, e *CIA*. Per maggiori chiarimenti sulle funzioni utilizzate appartenenti al modulo **SQLite**, si veda `docs.python.org/2/library/sqlite3.html`.

```
1  def sqlwiki(self):
2    conn=sqlite3.connect('dbpedia.db')
3    c=conn.cursor()
4    c.execute('''CREATE TABLE DBpedia (indice, link)''')
5    valori=[] #lista tuple
6    with open("DictWiki.txt") as f:
7      for x in f:
8        parola=x[:]
9        sito=x[:]
10       parola=parola.replace(',',' ')
11       parola=parola.replace('_',' ')
12       parola=parola.replace('-',' ')
13       parola=parola.replace('%',' ')
14       parola=parola.replace(':',' ')
15       parola=parola.replace('.',' ')
16       parola=parola.replace('/',' ')
17       parola=parola.replace('\n','')
18       parola=parola.replace('  ',' ')
19       parola=parola.replace('( ','(')
20       parola=parola.replace(' )',')')
21       valori.append((parola,sito))
22
23   c.executemany('INSERT INTO DBpedia VALUES (?,?)',valori)
24   conn.commit()
25   c.close()
26   conn.close()
```

[4]cit. `docs.python.org/2/library/sqlite3.html`

[5][...]è un sistema software progettato per consentire la creazione e la manipolazione [...] e l'interrogazione efficiente [...] di database [...]. (Wikipedia)

```
def sqlfree(self):
  conn=sqlite3.connect('freebase.db')
  c=conn.cursor()
  c.execute('''CREATE TABLE FreeBase (indice, link)''')
  valori=[] #lista tuple
  with open("DictFree.txt") as f:
    for x in f:
      campi=x.split(None,1)
      parola=campi[0]
      parola=parola.replace(',',' ')
      parola=parola.replace('_',' ')
      parola=parola.replace('-',' ')
      parola=parola.replace(':',' ')
      parola=parola.replace('.',' ')
      parola=parola.replace('%',' ')
      parola=parola.replace('/',' ')
      parola=parola.replace('\n','')
      parola=parola.replace('  ',' ')
      parola=parola.replace('( ','(')
      parola=parola.replace(' )',')')
      parola=parola.rstrip(' ')
      sito=campi[1]
      valori.append((parola,sito))
  c.executemany('INSERT INTO FreeBase VALUES (?,?)',valori)
  conn.commit()
  c.close()
  conn.close()
```

```
def sqlcia(self):
  conn=sqlite3.connect('ciagov.db')
  c=conn.cursor()
  c.execute('''CREATE TABLE CiaGov (indice, link)''')
  valori=[] #lista tuple
  with open("DictCia.txt") as f:
    for x in f:
      campi=x.split(None,1)
      parola=campi[1]
      sito=campi[0]
      parola=parola.replace('\n','')
      #li inserisco al contrario!
      valori.append((parola+' ',sito))

  c.executemany('INSERT INTO CiaGov VALUES (?,?)',valori)
  conn.commit()
  c.close()
  conn.close()
```

Per tutte e tre le funzioni ho creato un oggetto **connessione**, chiamato 'conn', che rappresenta il database che si vuole creare, e un oggetto **cursore**, chiamato 'c', di cui si

chiama il metodo *execute* passandogli un comando SQL. Il comando passato al metodo *execute* è inizialmente quello per la creazione della tabella, ossia 'CREATE TABLE', e poi per l'inserimento dei valori 'INSERT INTO'.

Le tabelle create rappresentano le entità e i relativi attributi, visti nel **Capitolo 2.2.1 Diagramma E/R**, e quindi contengono gli indici e i link per accedere ad ogni risorsa che descrivono.

In seguito alla creazione delle tabelle, ogni file TXT viene scandito per estrapolarne le variabili **parola** e **sito**, che contengono rispettivamente il valore che verrà inserito nella tabella come *indice* e quello che verrà inserito come *link*.

Il valore presente in **sito** verrà aggiunto tale e quale al database, in quanto andrà riportato uguale in fondo all'indirizzo Web del sito a cui vogliamo accedere per ricreare l'URL della risorsa cercata. Il valore presente in **parola**, invece, diventerà la chiave di ricerca del database. Esso è soggetto a molte modifiche per rimpiazzare quei caratteri che risulterebbero scomodi per effettuare delle ricerche sul database (es. _, :, etc.).

Con il metodo *executemany*, i valori **parola** e **sito** vengono infine inseriti nelle rispettive tabelle, nei rispettivi database.

Ora ho ottenuto i files in **Figura 3.4** contenenti rispettivamente i database di *CIA*, *DBpedia* e *Freebase*.

3.3 L'interrogazione

All'apertura del programma, viene chiesto all'utente di inserire una domanda. La lingua di inserimento è l'italiano e successivamente spiegherò com'è però possibile modificare la lingua di input in modo da poterla adattare alla lingua dell'utente che lo utilizza.

Nei sottocapitoli che seguiranno verranno illustrate le fasi del passaggio della domanda in italiano al sito **Google translate**, dell'interrogazione di *Wolframalpha* e dell'interrogazione dei database dei siti internet.

Quello che si vuole ottenere alla fine di questi passaggi sono una lista contenente i link per accedere alle risorse sul Web e una lista contenente le disambiguazioni possibili per tali risorse.

3.3.1 Traduzione con Google

L'obiettivo di questa prima fase è quello di tradurre la domanda fatta dall'utente dalla lingua d'origine all'inglese, in modo da poterla poi sottoporre al sito *Wolframalpha*, interamente in inglese.

Per ottenere ciò, ho creato una funzione in Python che, data una frase in ingresso, la traducesse automaticamente, utilizzando il sito `translate.google.com`.

Who is Barack Obama?

Figura 3.5: esempio di traduzione della domanda *Chi è Barack Obama?*

La funzione che utilizzo a tale scopo (**Figura 3.7**) è *translate*, che prende in ingresso la frase da tradurre e che viene invocata alla pressione del pulsante *Cerca*, dell'interfaccia grafica, con la chiamata mostrata nella figura seguente:

```
domanda=self.translate(domanda)
```

Figura 3.6: chiamata della funzione *translate*

All'inizio della funzione ho subito modificato la domanda che viene passata, ossia *to_translate*, per eliminarle o aggiungerle i caratteri che creerebbero problemi durante la traduzione con **google translate**.

Google translate, infatti, nonostante sia il miglior sito di traduzione online, presenta ancora diversi problemi legati alla traduzione di caratteri e forme di abbreviazione propri delle lingue, ad esempio, per la lingua italiana, l'inserimento di *é* al posto di *è*, oppure le abbreviazioni quali *cos'è* o *quand'è*. Per questo motivo ho deciso di risolvere manualmente questi problemi nei quali sono incappata, anche se ve ne sono altri che sicuramente saranno sfuggiti al mio controllo. Si invita quindi, nel caso di un utilizzo futuro del programma, di modificare in questo senso la domanda sottoposta in modo da ottenere una corretta traduzione.

Alla riga 2 della **Figura 3.7**, ho eliminato il carattere ' ", mentre nelle righe immediatamente successive, l'ho ripristinato tra le parole *dov* ed *è* e tra *cos* ed *è*, in quanto solamente in questo caso la mancanza del simbolo di apostrofo non permette la corretta traduzione della frase. Anche la direzione dell'apostrofo che accenta la lettera 'e' è stata corretta per gli stessi motivi.

Le variabili **to_language** e **language** identificano la lingua in cui e da cui tradurre la frase. È su quest'ultima che bisogna apportare delle modifiche se si vuole cambiare la lingua di input delle interrogazioni. Per farlo basta cambiare *'it'* e sostituirlo con la sigla della lingua dalla quale ci interessa tradurre. Inoltre è opportuno anche fare delle prove di interrogazioni e, laddove ci siano degli errori di traduzione, aggiungere dei metodi *replace* per modificare quei simboli o quelle forme linguistiche che non consentono una corretta traduzione dalla lingua scelta all'inglese.

Nella variabile **link** viene creata l'URL che consentirà di sottomettere la richiesta di traduzione a google translate (es. `http://translate.google.com/#it/en/chi%20%C3%A8%20Barack%20Obama%3F`), tramite il metodo *urllib2.Request*. Questo metodo consente di astrarre una richiesta URL; gli vengono passati una stringa che rappresenta una URL

```
1 def translate(self,to_translate):
2       to_translate=to_translate.replace('\'',' ')
3       to_translate=to_translate.replace('dov è','dov\'è')
4       to_translate=to_translate.replace('dov é','dov\'è')
5       to_translate=to_translate.replace('cos é','cos\'è')
6       to_translate=to_translate.replace('cos è','cos\'è')
7       to_langage='en'
8       langage='it'
9       agents = {'User-Agent':"Mozilla/4.0 (compatible; MSIE 6.0; Windows
  NT 5.1; SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR
  3.0.04506.30)"}
10      before_trans = 'class="t0">'
11      link = "http://translate.google.com/m?hl=%s&sl=%s&q=%s" %
  (to_langage, langage, to_translate.replace(" ", "+"))
12      request = urllib2.Request(link, headers=agents)
13      page = urllib2.urlopen(request).read()
14      result = page[page.find(before_trans)+len(before_trans):]
15      result = result.split("<")[0]
16
17      return result
```

Figura 3.7: funzione **translate**

valida, e uno o più *headers*, sottoforma di un dizionario, che rappresentano le opzioni dello *user agent*[6], in questo caso del browser Web *Mozilla*.

La pagina HTML corrispondente all'URL viene recuperata alla riga 13 con il metodo *urlopen.read()*. Ciò che ci viene restituito in questa pagina è la traduzione in inglese della frase che google ha effettuato.

Il passaggio successivo è quello di isolare dal resto del codice HTML la traduzione della nostra frase. Per farlo cerchiamo nel codice HTML una particolare frase, con il metodo *find*, alla riga 14, che si trova nella variabile *before_trans*, che sappiamo per certo si trovi immediatamente prima dell'inizio della frase tradotta.

Isolata la traduzione, essa viene 'ripulita' da ciò che la segue, ossia dal codice della pagina HTML che non ci interessa, e restituita al programma principale per essere utilizzata nel passaggio successivo, ossia nell'interrogazione di *Wolframalpha*.

3.3.2 Ottenere la risposta con Wolframalpha

In questa fase, ciò che abbiamo è la domanda in lingua inglese, che invieremo al sito *Wolframalpha* per ottenere una risposta.

La funzione che utilizzo in questa fase è mostrata nel codice sottostante. La chiamata di funzione viene fatta mediante un **thread**, istanziato e invocato dal programma prin-

[6]Lo user agent è un'applicazione installata sul computer dell'utente che si connette ad un processo server. (Wikipedia)

cipale tramite il metodo *threading.Thread*[7], al quale vengono passati come argomenti la frase in inglese e due code, istanziate mediante il metodo *Queue.Queue*[8], che consentono la comunicazione tra il thread e il programma principale.

```
1   def Wolframalpha(self,frase,q,q2):
2     URL='http://www.wolframalpha.com/input/?i='+frase
3     q.put(('Wolframalpha',URL))
4     sock=urllib.urlopen(URL)
5     htmlSource=sock.read()
6     sock.close()
7     if 'know␣how␣to␣interpret␣your␣input' in htmlSource or 'Using␣closest␣
        Wolfram|Alpha␣interpretation:' in htmlSource :
8       return 0
9     if 'Result<span␣class="colon">:</span></h2>' in htmlSource:
10      webbrowser.open(URL)
11
12      f=open('wolf.html','w')
13      f.write(htmlSource)
14      f.close()
15      indice=htmlSource.find('context.jsonArray.popups.pod_0200.push(␣{"
          stringified":␣')
16      result=htmlSource[indice:]
17      indice=result.find('}␣catch(e)␣{␣}')
18      result=result[:indice]
19      indice=result.find(':␣"')
20      indice=indice+3
21      result=result[indice:]
22      indice=result.find('"')
23      result=result[:indice]
24      #gestione lettere
25      result=result.replace('&iacute;','i')
26      result=result.replace('\\\'s','')
27
28      if '\\' in result:
29        indice=result.find('\\')
30        result=result[:indice]
31
32      if '(' in result:
33        indice=result.find('(')
34        result=result[:indice]
35
36      result=result.replace('&#39;','\'')
37      result=result.replace('&quot','')
38
39      risultatocapitale=''
40      if frase.find('capital')!=-1:
```

[7] http://docs.python.org/2/library/threading.html
[8] http://docs.python.org/2/library/queue.html

```
41          lista=result.split(',')
42          if len(lista)>=3:
43            risultatocapitale=str(lista[0])+','+str(lista[-1])
44
45        if risultatocapitale!='':
46          q2.put(risultatocapitale)
47        else:
48          q2.put(result)
49      else:
50        webbrowser.open(URL)
```

Una pecca di *Wolframalpha* è quella che non è in grado di rispondere a tutte le domande che gli vengono sottoposte. Il sito, infatti, potrebbe o non conoscere la risposta alla domanda fatta, o non comprendela, e in tal caso rispondere con un messaggio di avvertimento.

Per questo motivo viene inserito nel codice della funzione *Wolframalpha* un controllo sul codice HTML della pagina Web contenente la presunta risposta.

La suddetta pagina Web viene recuperata dal sito tramite il metodo già visto in precedenza, *urllib.urlopen*. Si controlla poi se in essa è presente almeno una delle frasi '*Doesn't know how to interpreter your input*' o '*Using closest Wolfram/Alpha interpretation*', in questo caso, allora, il sito non conosce la risposta o non riesce ad interpretare la domanda. Il thread termina senza riportare nulla al programma principale, il quale gestirà a tempo debito questa eccezione.

Diversamente da quanto appena detto, se nel codice HTML è invece presente una sezione '*Result*' allora è possibile che vi sia una risposta alla domanda fatta. Questo non è sempre vero in quanto, per alcune domande, *Wolframalpha* non conosce la risposta e, in tal caso, la sezione conterrà la frase '*(data not avaible)*'. Anche questa eccezione verrà gestita dal programma principale al termine dell'esecuzione del thread.

Se si presume ci sia una risposta, si cerca la sezione *Result* nel codice HTML e si isola. Il risultato ottenuto viene 'pulito' da caratteri scomodi, come i simboli o le lettere accentate (es. '*'*' invece di '*'*' o '*í*' al posto di '*ì*'). Inoltre, se nel risultato sono presenti le parentesi tonde, esse, con il loro relativo contenuto, vengono ignorate in quanto contengono un particolare aggiunto al risultato che, per le ricerche sui database che dobbiamo effettuare, risulterebbe più scomodo che utile.

Non meno degno di nota è il problema che sorge in presenza di risultati riguardanti le capitali mondiali. Se nella domanda è presente la parola '*capital*', ossia se sto chiedendo la capitale di un certo Stato, il risultato verrà ulteriormente scremato. Questo perchè, a questa domanda, *Wolframalpha* risponde fin troppo dettagliatamente.

Per esempio, alla domanda *'qual'è la capitale dell'irlanda?'*, Wolframalpha risponde 'correttamente', ossia indicando la capitale seguita dal nome dello Stato a cui appartiene, vedi **Figura 3.8**.

Può capitare, invece, che risponda in modo troppo prolisso alla domanda di capitali di altri Stati, come ad esempio l'Italia; alla domanda '*Qual'è la capitale d'Italia?*', risponde

Figura 3.8: esempio di risposta sul sito Wolframalpha

infatti inserendo anche il nome della regione nella quale si trova la capitale, oltre che allo Stato, vedi Figura seguente.

Figura 3.9: esempio di risposta sul sito Wolframalpha

Questo risultato sarebbe controproducente per l'interrogazione di un database, in quanto comporterebbe la ricerca di un'entità che contenga contemporaneamente tutte e tre le parole cercate nel suo attributo **indice**, ossia *Rome*, *Lazio* e *Italy*. In questo caso vengono mantenute solo le parole agli estremi del risultato, ossia la capitale e lo Stato.

Il risultato così modificato viene restituito al programma principale attraverso la *queue* e viene visualizzata la pagina Web su cui abbiamo appena terminato di lavorare, contenente la domanda e la relativa risposta.

Il passaggio successivo è basato sulla selezione delle parole chiave dal risultato ottenuto, per poterle usare come chiave di ricerca nei database.

3.3.3 Scrematura della risposta

Il risultato passato al programma principale viene ora analizzato per verificare se è nullo, ossia se non era presente su *Wolframalpha*, o se non era disponibile ('*(data not avaiable)*'). Se questo si verifica, viene presa la domanda fatta inizialmente come risposta, altrimenti si tiene quella fornita dal sito.

```
1     if result==''' or result=='(data␣not␣avaible)':
2       tokens= nltk.word_tokenize(domanda)
3     else:
4       tokens= nltk.word_tokenize(result)
```

La fase che analizzeremo ora è quella di scrematura del risultato o della domanda fino ad ottenere una lista di parole chiave da utilizzare per l'interrogazione dei database.

Lo strumento utilizzato in questa fase è **NLTK** (Natural Language Toolkit)[9], una suite che consente di analizzare e lavorare con le frasi in linguaggio naturale.

Utilizzando il metodo *nltk.word_tokenize*, mostrato nel codice sovrastante, effettuo una **tokenizzazione**[10], detta anche analisi lessicale, sulla risposta, ottenendo così una lista di lessemi.

Passando questa lista alla funzione *inizializza*, di cui ne è presente una parte nel codice sottostante, ne elimino le **stopwords**[11] inglesi, ottenendo così la lista delle parole chiave da utilizzare nell'interrogazione dei database.

```
1     self.wnl=nltk.WordNetLemmatizer()
2
3     for t in tokens:
4       t=t.lower()
5       if (not t in stopwords.words('english') and len(t)>1) or t=='|':
6         self.tokens.append(self.wnl.lemmatize(t))
```

Utilizzando il metodo *nltk.WordNetLemmatizer().lemmatize* sono in grado di effettuare anche lo **stemming**[12] di ogni parola chiave.

A questo punto quello che ho è la lista di parole chiave ridotte alla loro forma base estrapolate dalla risposta di *Wolframalpha* o dalla domanda dell'utente. Il passaggio che segue è quello vero e proprio dell'interrogazione dei database.

3.3.4 Interrogazione dei database

Questo passaggio, ossia il lavoro di interrogazione dei tre database, avviene concorrentemente. Questo perchè il programma principale, crea, arrivato a questo punto, tre threads, uno per ogni database da interrogare, che fa partire uno di seguito all'altro.

Il codice su cui lavorano i tre threads è abbastanza simile. Nelle funzioni sottostanti ho riportato per questo motivo solo il codice dell'interrogazione dei database di *DBpedia*

[9][...]is a leading platform for building Python programs to work with human language data. [...]a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning. (`nltk.org`)

[10]un token è un blocco di testo categorizzato, normalmente costituito da caratteri indivisibili chiamati lessemi. (Wikipedia)

[11]Stopwords are those words that have no discriminative meaning and often removed while learning classifiers. (Wikipedia)

[12]Lo stemming è il processo di riduzione della forma flessa di una parola alla sua forma radice, detta tema. (Wikipedia)

e di *CIA*, da parte dei relativi threads, in quanto quello di *Freebase* non si discosta di molto da quello di *DBpedia*. Il codice delle funzioni non è completo, ma mostra solamente la sezione riguardante l'interrogazione vera a propria dei database (per il codice completo si rimanda all' **Archivio dei codici**).

```
1  def DBpedia(self):
2    try:
3      conn = sqlite3.connect('dbpedia.db')
4      c = conn.cursor()
5    except IOError:
6      pass
7
8    ris=[]
9    probabile=[]
10   probabile2=[]
11   Wdic=[]
12
13   for i in self.__tokensnostop_:
14     n=len(i)
15     domanda='SELECT * FROM DBpedia WHERE indice LIKE ?'+' AND indice
         LIKE ?'*(n-1)
16     dati=[]
17     for h in i:
18       dati.append('%'+h.capitalize()+'%')
19     c.execute(domanda,dati)
20     probabile.append(c.fetchall())
21   c.close()
22   conn.close()
```

Come prima operazione creo una **connessione** con il database che mi interessa, utilizzando il metodo *sqlite3.connect*, e un oggetto **cursore**, per sottomettergli le operazioni (per maggiori informazioni sulla funzione di questi oggetti, rivedere il **Capitolo 3.2.2**).

La formulazione dell'interrogazione da sottoporre al database avviene come una normale query SQL: '*SELECT * FROM NomeTabella WHERE indice LIKE ?*'. Scelgo di selezionare tutte le informazioni delle entità il cui indice contiene la parola identificata da '*?*'.

L'utilizzo di '*?*' risulta di maggiore sicurezza ed è stato sostituito al metodo di sottomettere stringhe all'interrogazione (es. *SELECT * FROM stocks WHERE symbol= '%s' % symbol*). L'utilizzo di quest'ultime, infatti, renderebbe il programma vulnerabile ad attacchi SQL injection[13], dovuti all'inefficienza dei controlli sui dati ricevuti in input. Utilizzando '*?*', invece, i valori vengono passati direttamente al metodo *execute* tramite una tupla.

[13][...]è una tecnica dell'hacking mirata a colpire le applicazioni web che si appoggiano su un DBMS di tipo SQL. (Wikipedia)

Nello specifico caso di *DBpedia*, come si può vedere alla riga 18 del codice sovrastante, decido di selezionare solo i risultati che nell'indice contengono tutte le parole chiave ricavate nei passaggi precedenti. Per farlo interrogo il database passandogli una lista di parole e concatenandogli richieste SQL pari al numero delle parole chiave stesse.

Diversamente il caso di *CIA*, nel quale seleziono i risultati che contengono nell'indice o nel link una parola chiave alla volta. Questo perchè nel database cerco i nomi propri o i domini degli Stati, e non mi interessa quindi creare una combinazione di parole per raffinare la ricerca, come invece avviene nei database di *DBpedia* e di *Freebase*.

```
1     for t in self.__tokensnostop_:
2       for i in t:
3         c.execute('SELECT * FROM CiaGov WHERE indice LIKE ? OR link LIKE ?
              ',['%'+i.capitalize()+'%',i])
4         ris=c.fetchall()
```

Al termine dell'interrogazione otteniamo dai database una lista di risultati che, nel capitolo seguente, verranno analizzati e scremati per creare una lista dei risultati più idonei rispetto alla domanda fatta.

3.4 Risultati e disambiguazioni

In questo capitolo vedremo come i risultati ottenuti dall'interrogazione dei tre database, vengano scremati per ottenere il risultato da mostrare come pagina Web e la lista di disambiguazioni dello stesso da offrire all'utente, per ogni sito analizzato.

Data una domanda, è possibile sia approfondire la risposta direttamente su uno dei siti Web offerti, sia scegliere da una lista una disambiguazione di significato per la risposta ottenuta.

La pagina Web che viene aperta scegliendo l'approfondimento, è il risultato ritenuto più congruo rispetto alle parole chiave utilizzate nella ricerca sui database. La lista di disambiguazioni viene invece creata cercando tra i risultati quelli che hanno maggiormente la 'forma' di disambiguazioni, ossia, per la maggiorparte dei casi, quelli che presentano al loro interno i simboli '*( )*'.

L'algoritmo che regola la selezione di entrambe verrà illustrato nel sottocapitolo seguente.

3.4.1 La scrematura dei risultati

L'algoritmo che ho utilizzato per la scrematura dei risultati è presentato nello pseudocodice sottostante. Illustrerò solamente il funzionamento per il database di *DBpedia* in quanto quello di *Freebase* se ne discosta solamente per piccoli particolari, e quello di *CIA* ne riporta pari pari solo una piccolissima parte, in quanto, essendo i suoi risultati in quantità molto inferiore, non necessitano di così tanti passaggi.

```
for i in lista di liste di parole chiave:
  if lenght(i) = 1 then
     for x in listarisultati:
        if lenght(x) = lenght(i) then
           listarisultatifinali ← x
        if x inizia con i(maiuscola) + '(' or 'The' +
         i(maiuscola) + '(' then
           listarisultatifinali ← x
  else
     listatokens ← i
for x in listarisultati:
  if x non contiene 'File' or 'Category' or 'Template' then
     for t in listatokens:
        cont ← 0
        contmax ← lenght(t)
        for i in t:
           l ← lenght(i)
           if x contiene i(maiuscola)+' ' or i+')' or
           '('+i or i(maiuscola)+')' or
            '('+i(maiuscola) then
              cont++
           else
              lista ← x
              for t in lista:
                 if lenght(t) = l then
                    cont++
         if cont = contmax then
            listarisultati2 ← x
```

Nella prima metà dello pseudocodice sovrastante, vi è una prima scansione di 'lista di liste di parole chiave'. Questo valore è una lista contenente tante liste quanti sono i risultati offerti da *Wolframalpha*.

Il sito, infatti, può restituire anche una lista di molteplici risultati dei quali, però, ho deciso di tenerne, nel caso ce ne siano più di uno, solo i primi due, per una questione di velocità dell'applicazione. Ogni risultato di *Wolframalpha* impiega infatti molto tempo a scandire l'intera lista dei risultati offerti dai database per verificare i criteri di accettazione o meno dell'elemento. Nel caso che questo passaggio si dovesse ripetere più di due volte, i tempi di attesa sarebbero inaccettabili per l'applicazione che voglio ottenere.

Il controllo sulla lunghezza della lista nella seconda riga, serve per differenziare la ricerca che applico sui risultati.

Se la lunghezza della lista è 1, ossia se la parola chiave da cercare è una sola, mantengo solo quei risultati che presentano la parola chiave tale e quale o che la presentano seguita da una o più parole tra parentesi. Questo perchè preparo già i risultati che mi serviranno nella successiva fase, quella della disambiguazione. Se esiste un risultato che coincide perfettamentente con la parola chiave cercata, sarà il risultato da mostrare come pagina Web del sito interrogato. Se ci sono risultati uguali ma seguiti dalle parentesi tonde e/o preceduti dalla parola '*The*', queste saranno le possibili disambiguazioni relative a quel risultato.

Se la lunghezza della lista è invece maggiore di 1, la stessa viene accodata in una lista chiamata *listatokens* che si utilizzerà per la scrematura dei risultati nei quali deve essere presente più di una parola chiave. Questo passaggio ci è utile, inoltre, per creare una lista di risultati nei quali non siano più presenti gli stessi di lunghezza 1, ossia quelli già analizzati nell'*if* precedente.

Nel secondo ciclo for, all'undicesima riga, ignoro tutti quei risultati offerti dai database che contengono le parole 'File', 'Category' e 'Template', in quanto contengono il link a quelle pagine Web che sono immagini o categorie di risultati, e quindi non utili per il mio scopo.

Avviene poi qui un primo controllo sui risultati, essi vengono scanditi e viene controllato se alcune delle parole chiave sono contenute all'interno di parentesi o se sono seguite da uno spazio. Se ciò avviene il risultato ci va bene, e incremento una variabile contatore che conta il numero di parole chiave che vengono trovate all'interno del risultato attuale tra quelli offerti dalla ricerca nel database.

Può invece capitare che il risultato offerto contenga sì la parola chiave cercata, ma non seguita da uno spazio o all'interno di parentesi. Questo è il caso di parole contenute all'interno di altre parole, ossia una parola composta (es. *king* e *kingdom*), che per noi è un risultato sbagliato e quindi da scartare. Do però una seconda possibilità ai risultati che rientrano in questa categoria, scegliendo di confrontare la lunghezza della parola attuale con quella della parola chiave e, nel caso coincidano, considerare ugualmente il risultato come valido.

Alla fine, solo i risultati in cui compaiono tante parole quanto il numero delle parole chiave vengono tenuti come risultati idonei e passano alla seconda fase di analisi per scremarsi ulteriormente.

```
for a in listatokens:
  nparole ← lenght(a)
  for i in listarisultati2:
    nparole2 ← lenght(i)
    if i contiene '(' then
      conto ← 0
      paroletraparentesi ← parole tra '(' e la fine di i
      for parole in a:
```

```
        if paroletraparentesi contiene parole or parole(maiuscolo) then
          conto++
      nparole2 ← nparole2-lenght(paroletraparentesi)+conto
      senzaparentesi ← parole dall'inizio di i fino a '('
    for t in senzaparentesi:
      if t in stopwords inglesi then
        nparole2--
    if nparole2 = nparole then
      listarisultati3 ← i
lung ← maxint
for i in listarisultati3:
  lungI ← lenght(i)
  if lungI < lung then
    lung ← lungI
    tupla ← i
if tupla diverso da vuota then
  listarisultatifinali ← tupla
for i in listarisultati3:
  listarisultatifinali ← i
```

Qui, vedi pseudocodice sovrastante, le parole chiave vengono nuovamente scandite e per ogni risultato nella lista creata precedentemente, viene controllato se esso contiene il simbolo '(. Se ciò si verifica, parte un controllo per verificare che non vi siano delle parole chiave presenti tra le parentesi e, nel caso ce ne siano, vengono conteggiate, altrimenti le parole tra le parentesi vengono semplicemente ignorate in questa fase di scrematura, in quanto contenenti informazioni non utili.

```
for a in listatokens:
  nparole ← lenght(a)
  for i in listarisultati2:
    senzaparentesi ← i
    nparole2 ← lenght(i)
    if i contiene '(' then
      conto ← 0
      paroletraparentesi ← parole tra '(' e la fine di i
      for parole in a:
        if paroletraparentesi contiene parole or parole(maiuscolo) then
          conto++
      nparole2 ← nparole2-lenght(paroletraparentesi)+conto
      senzaparentesi ← parole dall'inizio di i fino a '('
    for t in senzaparentesi:
```

```
      if t in stopwords inglesi then
        nparole2--
    if nparole2 = nparole:
      listarisultati3 ← i

lung ← sys.maxint
for i in listarisultati3:
  lungI ← lenght(i)
  if lungI < lung then
    lung ← lungI
    tupla ← i
if tupla diverso da vuota then
  listarisultatifinali ← tupla
for i in listarisultati3:
  listarisultatifinali ← i
```

La parte di risultato privata delle parentesi, viene scandita e vengono eliminate le stopwords presenti dal conteggio finale del numero di parole. Infine, solo i risultati con un numero di parole pari al numero delle parole chiave vengono mantenuti.

Questo passaggio è molto selettivo ma è stato mantenuto in quanto l'obbiettivo finale è quello di offrire solamente le risposte più corrette e che si avvicinano maggiormente a ciò che abbiamo chiesto.

L'ultima parte dell'algoritmo inizia alla riga 20 dello pseudocodice sovrastante e rappresenta il metodo di scelta dell'URL univoca da restituire come migliore risultato per il sito Web analizzato. La risposta migliore viene scelta in base alla sua lunghezza, intesa come numero di caratteri.

Il risultato con il minor numero di caratteri viene considerato dall'algoritmo quello che più si avvicina alla richiesta effettuata e viene restituito al programma principale già sottoforma di URL. Gli altri risultati vengono invece inseriti in una lista che diventerà poi la lista delle disambiguazioni disponibili relative al suddetto risultato del relativo sito Web.

3.4.2 Le disambiguazioni

Il programma principale, una volta che sono ritornati tutti i threads addetti all'interrogazione dei database, consente all'utente di approfondire su un altro sito Web la risposta ottenuta, oppure di disambiguare quella visualizzata.

Nella disambiguazione, il programma principale interroga la classe addetta alla scrematura dei risultati per farsi resistuire la lista di tutti i risultati possibili, comprendente anche il risultato migliore già mostrato sulla pagina Web. Alla scelta della disambiguazione corretta il programma apre la pagina Web relativa.

La qualità e la quantità delle disambiguazioni proposte varia in base alla domanda chè è stata sottoposta: se la parola chiave da ricercare nei database è solo una, come nell'esempio mostrato in figura, le disambiguazioni possibili saranno molteplici; se invece già la risposta alla domanda sottoposta è molto specifica, i risultati saranno molto ridotti, se non addirittura, non offriranno nessuna disambiguazione. In questo caso spetta all'utente modificare la domanda per renderla più specifica, o interrogare il programma utilizzando direttamente la risposta che ha ottenuto dal sito di *Wolframalpha*, al fine di 'aiutare' il programma stesso ad offrire i risultati migliori.

3.5 GUI

Infine voglio mostrare brevemente il funzionamento dell'interfaccia grafica che ho creato. Alla prima apertura, dopo il controllo della presenza dei file di configurazione, in caso della loro mancanza, viene visualizzata una schermata di avvertimento, vedi **Figura 3.10**. Se il controllo è andato, invece, a buon fine, la finestra principale che viene mostrata all'utente quella in **Figura 3.11**. Alla pressione del pulsante **Cerca**, si avvia la fase di traduzione seguita da quella di interrogazione di *Wolframalpha* e dei database.

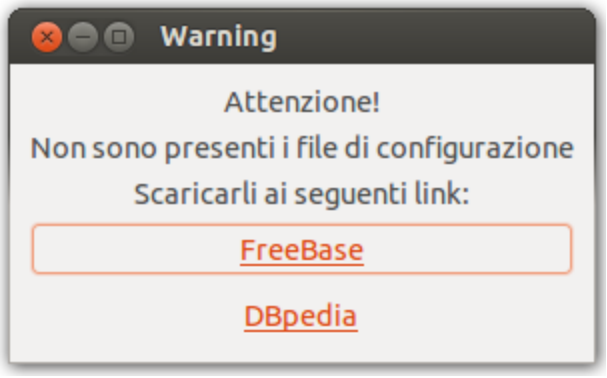

Figura 3.10: Prima apertura del programma

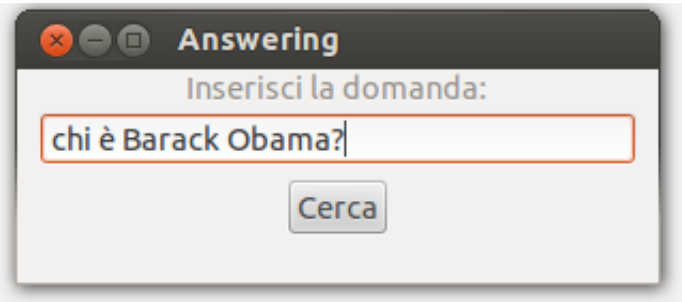

Figura 3.11: esempio di interrogazione

Nel codice sottostante si vede riportata una piccola parte delle funzione *cerca*, avviene prima un controllo sui risultati restituiti dai threads adibiti all'interrogazione dei database.

```
1     if 'Freebase' in lista and lista['Freebase']!='':
2       self.listadivisita.append(('Freebase',lista['Freebase']))
3       if self.indicevisita==-1:
4         self.builder.get_object('approfondisci').set_uri(self.
              listadivisita[self.indicevisita][1])
5         self.builder.get_object('boxapprofondisci').show()
6         self.builder.get_object('WindowMain').resize(300,150)
```

```
7         self.indicevisita=self.indicevisita+1
8
9       self.indicevisita=self.indicevisita+1
10      self.listadivisita.append(('Wolframalpha',lista['Wolframalpha']))
11      self.indicemax=self.indicevisita
12      self.indicevisita=0
13
14      butt=self.builder.get_object('approfondisci')
15      butt.set_uri(self.listadivisita[self.indicevisita][1])
16      label='Approfondisci su \n'+self.listadivisita[self.indicevisita][0]
17      butt.set_label(label)
18
19      del lista
20      if risultato==False and len(self.listadivisita)==0:
21        self.builder.get_object('mexwarning').set_text('Nessun risultato
            disponibile!')
22        self.builder.get_object('boxavvertimento').show()
```

Al pulsante relativo all'approfondimento viene collegata l'URI corretta e vengono popolate le variabili necessarie al funzionamento dell'interfaccia.

Alla fine, alla riga 20, viene controllata la presenza di almeno un risultato, e, nel caso non ci sia, viene implementata la gestione del caso.

L'interfaccia si modifica poi nella **Figura 3.12**, ora è possibile sia disambiguare attraverso l'interfaccia il risultato ottenuto, premendo il pulsante '*Intendevi altro?*', sia visualizzare la risposta sul sito Web successivo, alla pressione del pulsante '*Approfondisci su ...*'.

Alla pressione del pulsante di approfondimento gli indici di visita vengono aggiornati e il nome del pulsante e il relativo link vengono modificati con quelli successivi, in attesa della prossima pressione (vedi codice sottostante).

```
1       if self.indicevisita==self.indicemax:
2         self.indicevisita=0
3       else:
4         self.indicevisita=self.indicevisita+1
5
6       url=self.listadivisita[self.indicevisita][1]
7       butt=self.builder.get_object('approfondisci')
8       butt.set_uri(url)
9       label='Approfondisci su \n'+self.listadivisita[self.indicevisita][0]
10      butt.set_label(label)
```

La lista di disambiguazioni che vengono mostrate sono quelle ottenute nella fase di scrematura dei risultati, illustrata nel **Capitolo 3.4.1**. Alla pressione del pulsante di disambiguazione avviene un controllo su quale risultato bisogni disambiguare e se, nel caso, esistano o meno disambiguazioni disponibili (vedi il codice sottostante), dopodiché la lista viene mostrata nell'interfaccia grafica.

```
1         if 'freebase' in self.dis:
```

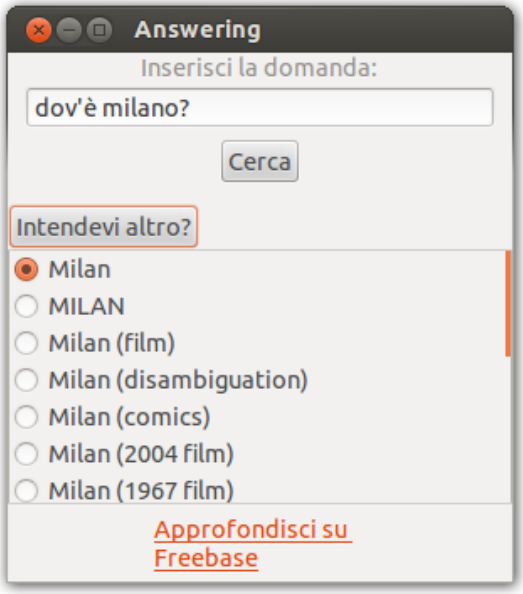

Figura 3.12: esempio di funzionamento del programma

```
2            if len(self.dis['freebase'])<=1:
3              nodisambigua=True
4            else:
5              self.boxfree=self.crearadio(self.dis,'freebase')
6              self.window.add(self.boxfree)
7          else:
8            nodisambigua=True
9
10       if nodisambigua:
11         self.builder.get_object('mexwarning').set_text('Nessuna␣
             disambiguazione␣disponibile')
12         self.builder.get_object('boxavvertimento').show()
13         return 0
14       else:
15         self.scrolledshow=True
16         window2.add(self.window)
17         window2.show_all()
```

Solamente dopo la scelta di una disambiguazione differente da quella impostata dal programma come risposta più corretta, e, quindi, da quella presente nel link del pulsante di approfondimento, è possibile aprirne la pagina Web, grazie alla comparsa del pulsante '**Vai**' (**Figura 3.13**).

Quindi, il pulsante 'Vai', apre l'URL della disambiguazione sul sito Web corrente mentre 'Approfondisci su' apre l'URL della risposta sul successivo sito Web che avrà le sue disambiguazioni diverse da quelle mostrate ora.

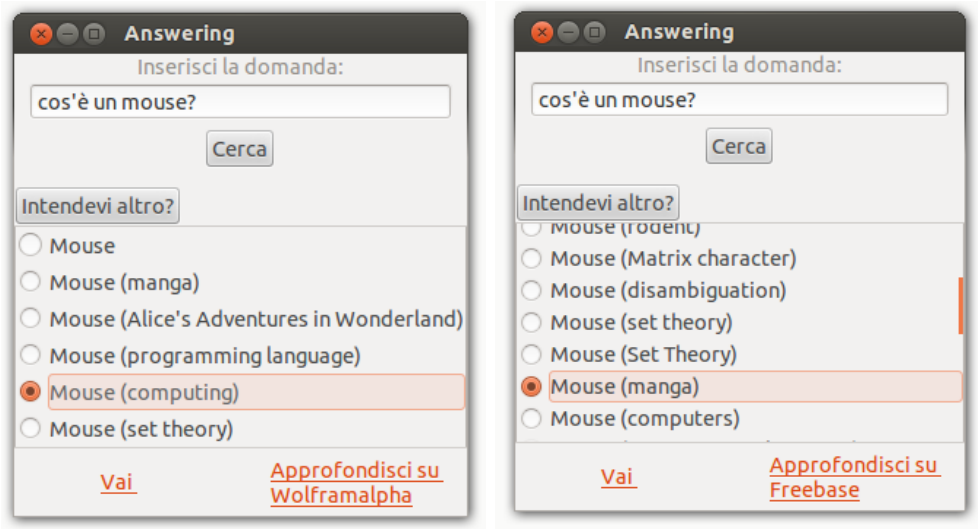

Figura 3.13: esempi di disambiguazione

Alla pressione di una scelta di disambiguazione, compare il pulsante 'Vai', che contiene il link alla risorsa scelta (vedi codice sottostante).

```
1    if button.get_active():
2      state='on'
3      if sito=='cia':
4        self.builder.get_object('Vai').set_uri('https://www.cia.gov/
           library/publications/the-world-factbook/geos/'+scelta+'.html')
5      if sito=='dbpedia':
6        self.builder.get_object('Vai').set_uri('http://dbpedia.org/
           resource/'+scelta)
7      if sito=='freebase':
8        self.builder.get_object('Vai').set_uri('http://www.freebase.com/'+
           scelta)
9      self.builder.get_object('Vai').show()
10   else:
11     state='off'
```

Per il codice completo si rimanda all'**Archivio dei codici**, in questo capitolo sono state presentate solamente le implementazioni più significative per il funzionamento del programma.

Capitolo 4

Analisi dei risultati

Conlusa la spiegazione del funzionamento del programma, voglio qui analizzare i risultati ottenuti attraverso una tabella che vuole essere un coinciso riassunto della bontà degli stessi. Le interrogazioni che mostro in essa sono quelle più significative e che meglio esprimono la qualità del programma creato.

Questo riassunto raccoglie solo una piccolissima parte di tutte le interrogazioni che sono state effettuate durante la realizzazione dell'applicazione; di queste, ho cercato di riportarne il più possibile in riferimento ad argomenti differenti, che spaziano quindi dalla geografia, all'informatica, all'attualità, etc..

Nella tabella sono presentate 10 interrogazioni sottoposte all'applicazione e i relativi risultati ottenuti.

L'elenco delle interrogazioni, divise per argomento, è il seguente:

- **Geografia.**
  Interrogazione 1: 'Qual'è la capitale dell'Australia?'
  Interrogazione 2: 'Canberra'

- **Storia.**
  Interrogazione 3: 'Chi ha scoperto l'America?'

- **Matematica.**
  Interrogazione 4: '7*14+9-157'

- **Attualità.**
  Interrogazione 5: 'Chi è il presidente tedesco?'

- **Informatica.**
  Interrogazione 6: 'Cos'è un proxy?'
  Interrogazione 7: 'Cos'è Java?'

- **Curiosità.**
  Interrogazione 8: 'Cos'è un mouse?'
  Interrogazione 9: 'Cos'è una FIAT?'
  Interrogazione 10: 'Cos'è Max Payne?'

I risultati ottenuti in seguito alla sottoposizione delle risposte sono organizzati in righe, alle quali viene assegnata una V per la presenza del risultato, X per la sua mancanza. Ulteriori approfondimenti vengono elargiti in seguito. Le righe contengono i seguenti valori, abbreviati per questioni di spazio:

- **Risposta Wolframalpha:** se è presente o meno una risposta alla domanda fatta. Per la precisione si intende la presenza di una sezione *Result* contenente una risposta vera e propria, e non una pagina esplicativa dell'entità cercata.
- **Disambiguazioni Wolframalpha:** se è possibile o meno scegliere una disambiguazione per la risposta ottenuta.
- **Risposta CIA:** se il sito della CIA è in grado di offrire un link ad una delle sue pagine.
- **Disambiguazioni CIA:** se è possibile scegliere tra più possibili Stati mondiali, equamente probabili.
- **Risposta DBpedia:** se il sito DBpedia è in grado di rispondere alla domanda sottomessa.
- **Disambiguazioni DBpedia:** se ci sono più risultati tra cui scegliere.
- **Risposta Freebase:** se il sito è in grado di offrire almeno una risposta.
- **Disambiguazioni Freebase:** se sono presenti disambiguazioni.

	1	2	3	4	5	6	7	8	9	10
Risp. Wolframalpha	*V*	*X*	*X*	*V*	*X*	*X*	*X*	*X*	*X*	*X*
Dis. Wolframalpha	*X*	*X*	*V*	*X*	*X*	*X*	*V*	*V*	*V*	*X*
Risp. CIA	*V*	*X*	*X*	*X*	*X*	*X*	*X*	*X*	*X*	*X*
Dis. CIA	*X*	*X*	*X*	*X*	*X*	*X*	*X*	*X*	*X*	*X*
Risp. DBpedia	*V*	*V*	*X*	*V*	*V*	*V*	*V*	*V*	*V*	*V*
Dis. DBpedia	*X*	*V*	*X*	*V*	*X*	*V*	*V*	*V*	*V*	*V*
Risp. Freebase	*X*	*V*	*X*	*X*	*X*	*V*	*V*	*V*	*V*	*V*
Dis. Freebase	*X*	*X*	*X*	*X*	*X*	*V*	*V*	*V*	*V*	*V*

Ho deciso di suddividere graficamente la tabella in base alla categoria delle interrogazioni che sottopongo alla mia applicazione per meglio analizzarne i risultati.

Le prime due interrogazioni sono state formulate diversamente per mostrare al lettore la differenza dei risultati che è possibile ottenere, facendo una domanda o inserendo solamente una parola chiave. Con l'interrogazione n°1 ottengo una sezione 'Result' da *Wolframalpha* contenente la capitale australiana che mi consente di avere una risposta anche sul sito della *CIA* in quanto in essa è presente anche lo stato di appartenenza della stessa. *DBpedia* risponde alla domanda ma non offre però disambiguazioni. Al contrario, con la seconda interrogazione, non ottengo nessuna risposta da *Wolframalpha* e da *CIA* ma in compenso ho un risultato da *Freebase* e più di uno da *DBpedia*.

L'interrogazione 3, quella di storia, da dei problemi nel recupero di risultati in quanto *Wolframalpha* non offre una risposta coincisa, diversamente da quanto ci si aspetterebbe. Questo è infatti un caso particolare per il quale le disambiguazioni di *Wolframalpha* tornano utili per riformulare meglio l'interrogazione aiutandosi con le disambiguazioni che lei stessa offre. Per questo esempio l'interpretazione che viene fatta della domanda in input è '*first humans reach North America via Beringia*', mentre quello che volevamo ottenere era una delle possibili disambiguazioni, ossia '*Christopher Columbus reaches the New World*'. Di conseguenza, non essendoci una sezione 'Result', vengono selezionate le parole chiave della domanda per interrogare le knowledge base di cui disponiamo che però non riescono a recuperare delle risposte.

L'interrogazione matematica (colonna n°4) è stata fatta in quanto *Wolframalpha* funziona anche da calcolatrice e si nota come, nonostante il risultato numerico ottenuto, anche *DBpedia* offra delle possibili risposte: 725 come prima risposta di cui offre un elenco di tutte le informazioni che possiede riguardanti questo numero; 700 come disambiguazione e tutte le informazioni dello stesso.

Una particolarità che ho notato analizzando i risultati ottenuti è il comportamento di *Freebase*. Questo sito, infatti, non riesce a rispondere a tutte le interrogazioni a cui invece offre una risposta *DBpedia*; questo è dovuto al fatto che la sua knowledge base è più ridotta e ogni MID corrisponde ad un concetto espresso attraverso una sola parola chiave che, spesso, nella knowledge base di DBpedia, è invece rappresentato da più parole

chiave e viene preferito dal mio algoritmo in quanto maggiormente specifico e quindi, si ipotizza, più corretto.

Sulla base di tutti i risultati analizzati e qui non riportati, posso concludere dicendo che l'applicazione da me realizzata risponde correttamente ad una grande quantità di domande generiche di qualsiasi argomento, mentre si trova maggiormente in difficoltà per quelle domande più complesse, o che sono molto specifiche, o a cui segue una lista di risposte, come: 'Quali sono stati i re di Francia?'; 'Quali libri ha scritto Stephen King?', 'Dove vive Jack Nicholson?', etc..

Questa applicazione ha anche una caratteristica, non volantariamente voluta, che è quella di far conoscere all'utente, tramite le disambiguazioni che offre, significati dei concetti cercati che, molti dei quali, magari, non si sarebbero mai scoperti.

Conclusioni

Concluso il lavoro di realizzazione della tesi, voglio dedicare questo capitolo all'esposizione degli obiettivi raggiunti, preposti all'inizio del lavoro.

Il lavoro di tesi che ho svolto è riassuntivo di un percorso di lavoro nel quale non sono mancate difficoltà e scelte fra diverse opzioni implementative, alcune delle quali sono poi state abbandonate a discapito di quelle utilizzate attualmente.

Voglio nuovamente sottolineare che, comunque, tutti gli obiettivi esposti nell'introduzione sono stati realizzati con ottimi risultati: a cominciare dall'estrazione dei dataset, per finire con la realizzazione dell'interfaccia grafica. Punto focale del mio lavoro era quello di realizzare un algoritmo per il Question Answering e ho risolto il tutto realizzando un programma che, interrogando diverse basi di conoscenza di siti Web eterogenei, riuscisse a dare una risposta precisa a ciò che era stato domandato, scontrandosi con problemi di traduzione e disambiguazione.

Detto ciò sono ancora presenti dei passaggi che potebbero essere ampliati in futuro:

- Il numero di knowledge base utilizzate: per ora si limita a tre, ma in presenza di più fonti informative sicure da cui crearne il relativo database da interrogare, la qualità e la quantità delle risposte si incrementerebbe notevolmente.

- Il sito *Wolframalpha* per la sottomissione di query: nonostante sia uno dei migliori siti per il question answering, è ancora in fase di ampliamento e miglioramento e la quantità di domande a cui riesce a rispondere è ancora limitata e non spazia per le domande più complesse.

- La traduzione delle domande: Google Translate compie ancora errori di traduzione che andrebbero minuziosamente rivisti e corretti dal mio programma non solo per quanto riguarda la lingua italiana, ma anche per, almeno, le altre lingue più comuni.

- L'interfaccia grafica: sicuramente migliorabile dal punto di vista estetico.

Elenco, infine, ciò che sono, invece, i punti di forza del programma da me realizzato:

- Possibilità di inserimento delle domande in qualsivoglia lingua e traduzione automatica delle stesse.

- Creazione automatica dei database e possibilità di ampliamento degli stessi in modo rapido.
- Scrematura efficiente e selezione precisa del miglior risultato da mostrare tra quelli ottenuti dall'interrogazione dei database.
- Selezione dei risultati idonei per la disambiguazione.
- Interfaccia di facile utilizzo utilizzabile da qualsiasi tipologia di pubblico, in termini di conoscenze pregresse.

Archivio dei codici

Di seguito verranno riportati tutti i codici scritti per il funzionamenteo del programma, di cui si sono viste le parti più salienti nel corso della tesi.

A.1 main.py

```
1 from interfaccia import Gui
2 import threading
3 from gestione.Creazionedizionari import Dizionari
4 from gi.repository import Gtk ,GObject, Gdk
5 import sys
6 import time
7
8 GObject.threads_init()
9
10 creato=False
11 ogg=Dizionari()
12
13 class Attesa(threading.Thread):
14   def __init__(self,label,lista,progressbar,win):
15     super(Attesa,self).__init__()
16     self.label=label
17     self.window=win
18     self.progressbar=progressbar
19     self.quit=False
20     self.lista=lista
21     self.activity_mode = True
22
23   def update(self):
24     for t in self.lista:
25       if not t.isAlive():
26         i=self.lista.index(t)
27         del self.lista[i]
28     if not self.lista:
29       self.quit=True
30       self.label.set_text("Creazione terminata!")
31       new_value = self.progressbar.get_fraction() +1
```

```
32        if new_value > 1:
33          new_value = 0
34        self.progressbar.set_fraction(new_value)
35        Gtk.main_quit()
36        self.window.hide()
37      else:
38        self.progressbar.pulse()
39      return False
40
41    def run(self):
42      while not self.quit:
43        GObject.idle_add(self.update)
44        time.sleep(0.1)
45
46  if __name__=='__main__':
47    #controllo presenza file contenenti i dati necessari
48    try:
49      open('page_ids_en.ttl')
50      open('freebase_links.nt')
51    except IOError:
52      #mostra i link da cui scaricare i file necessari.
53      creato=True
54      window=Gtk.Window(title="Warning")
55      window.set_position(Gtk.WindowPosition.CENTER)
56      window.set_border_width(10)
57      vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
58      window.add(vbox)
59      label1=Gtk.Label('Attenzione!')
60      label2=Gtk.Label('Non␣sono␣presenti␣i␣file␣di␣configurazione')
61      label3=Gtk.Label('Scaricarli␣ai␣seguenti␣link:')
62      butt1=Gtk.LinkButton('http://downloads.dbpedia.org/3.8/links/
          freebase_links.nt.bz2','FreeBase')
63      butt2=Gtk.LinkButton('http://downloads.dbpedia.org/3.8/en/page_ids_en.
          ttl.bz2','DBpedia')
64
65      vbox.pack_start(label1,True,True,0)
66      vbox.pack_start(label2,True,True,0)
67      vbox.pack_start(label3,True,True,0)
68      vbox.pack_start(butt1,True,True,0)
69      vbox.pack_start(butt2,True,True,0)
70
71      window.connect("delete-event", Gtk.main_quit)
72      window.connect("destroy", lambda _: Gtk.main_quit())
73      window.show_all()
74      Gtk.main()
75      exit(0)
76
77    mancano=False
78    try:
```

```
    #controllo presenza file txt contenenti gli elementi da inserire nei
        databse
    Dwiki=open('DictWiki.txt')
    Dfree=open('DictFree.txt')
    Dcia=open('DictCia.txt')
  except IOError:
    #solitamente eseguito solo alla prima apertura del programma
    mancano=True

  if mancano:
    #comparsa finestra di attesa
    window=Gtk.Window(title="Waiting...")
    window.set_position(Gtk.WindowPosition.CENTER)
    window.set_border_width(10)
    vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
    window.add(vbox)
    label=Gtk.Label('Creazione dizionari...')
    progressbar = Gtk.ProgressBar()
    vbox.pack_start(label,True,True,0)
    vbox.pack_start(progressbar, True, True, 0)
    window.connect("delete-event", Gtk.main_quit)
    window.connect("destroy", lambda _: Gtk.main_quit())
    window.show_all()

    lista=[]

    #apertura di vari thread che eseguono creazionedizionari.py
    try:
      Dwiki=open('DictWiki.txt')
    except IOError:
      t1=threading.Thread(target=ogg.Dwiki)
      t1.start()
      lista.append(t1)
      creato=True

    try:
      Dfree=open('DictFree.txt')
    except IOError:
      t1=threading.Thread(target=ogg.Dfree)
      t1.start()
      lista.append(t1)
      creato=True

    try:
      Dcia=open('DictCia.txt')
    except IOError:
      t1=threading.Thread(target=ogg.Dcia)
      t1.start()
      lista.append(t1)
```

```
127         creato=True
128
129       #thread per la gestione della finestra di attesa
130       t = Attesa(label,lista,progressbar,window)
131       t.start()
132
133       Gtk.main()
134
135     #uscita al termine della creazione.
136     if mancano:
137       #Reso necessario dall'uso di troppa memoria.
138       exit(0)
139
140     mancano=False
141     try:
142       #controllo ed eventuale creazione dei database
143       open('dbpedia.db')
144       open('freebase.db')
145       open('ciagov.db')
146     except IOError:
147       mancano=True
148
149     if mancano:
150       window=Gtk.Window(title="Waiting...")
151       window.set_position(Gtk.WindowPosition.CENTER)
152       window.set_border_width(10)
153       vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
154       window.add(vbox)
155       label=Gtk.Label('Creazione␣database...')
156       progressbar = Gtk.ProgressBar()
157       vbox.pack_start(label,True,True,0)
158       vbox.pack_start(progressbar, True, True, 0)
159       window.connect("delete-event", Gtk.main_quit)
160       window.connect("destroy", lambda _: Gtk.main_quit())
161       window.show_all()
162
163       lista=[]
164
165       try:
166         open('dbpedia.db')
167       except IOError:
168         t1=threading.Thread(target=ogg.sqlwiki)
169         t1.start()
170         lista.append(t1)
171         creato=True
172         label.set_text('Creazione␣database␣DBpedia')
173
174       if creato==False:
175         try:
```

```
176          open('freebase.db')
177        except IOError:
178          t1=threading.Thread(target=ogg.sqlfree)
179          t1.start()
180          lista.append(t1)
181          creato=True
182          label.set_text('Creazione database Freebase')
183      if creato==False:
184        try:
185          open('ciagov.db')
186        except IOError:
187          t1=threading.Thread(target=ogg.sqlcia)
188          t1.start()
189          lista.append(t1)
190          creato=True
191          label.set_text('Creazione database CiaGov')
192
193      t = Attesa(label,lista,progressbar,window)
194      t.start()
195
196      Gtk.main()
197
198    if creato:
199      exit(0)
200
201    #esecuzione dell'interfaccia grafica principale
202    Gui.gui()
```

A.2 Gui.py

```
1  import urllib,urllib2
2  from urllib2 import HTTPError
3  import sys
4  import nltk
5  import Queue
6  from gi.repository import Gtk ,GObject
7  import webbrowser
8  from gestione.Sceltasito import sceltasito
9  import threading
10 import time
11
12 GObject.threads_init()
13
14 class Attesa(threading.Thread):
15   def __init__(self,label,lista,progressbar,window):
16     super(Attesa,self).__init__()
17     self.label=label
18     self.progressbar=progressbar
```

```
19      self.quit=False
20      self.lista=lista
21      self.activity_mode = True
22      self.window=window
23
24    def update_label(self):
25      for t in self.lista:
26        if not t.isAlive():
27          i=self.lista.index(t)
28          del self.lista[i]
29      if not self.lista:
30        self.quit=True
31        self.label.set_text("Ricerca conclusa!")
32        new_value = self.progressbar.get_fraction() +1
33        if new_value > 1:
34          new_value = 0
35        self.progressbar.set_fraction(new_value)
36        Gtk.main_quit()
37        self.window.hide()
38      else:
39        self.progressbar.pulse()
40      return False
41
42    def run(self):
43      while not self.quit:
44        GObject.idle_add(self.update_label)
45        time.sleep(0.1)
46
47  class gui:
48    builder=None
49    def __init__(self):
50      self.indicevisita=-1
51      self.listadivisita=[]
52      self.indicemax=0
53      self.scrolledshow=False
54      self.primavolta=True
55      self.cercato=False
56
57      self.q=Queue.Queue()
58      self.ogg=sceltasito()
59
60      self.builder=Gtk.Builder()
61      self.builder.add_from_file('gui.glade')
62      dic={'on_windowMain_destroy':self.quit,'on_buttonCerca_clicked':self.
          cerca,'on_buttonDisambigua_clicked':self.disambigua,'
          on_buttonApprofondisci_clicked':self.approfondisci,'
          on_buttonVai_clicked':self.Vai}
63      self.builder.connect_signals(dic)
64      window=self.builder.get_object('WindowMain')
```

```
65      window.set_title("Answering")
66      window.connect("delete-event",Gtk.main_quit)
67      window.set_position(Gtk.WindowPosition.CENTER)
68      window.show()
69      Gtk.main()
70
71    def quit(self,widget):
72      sys.exit(0)
73
74    def crearadio(self,dis,sito):
75      box=Gtk.Box(orientation=Gtk.Orientation.VERTICAL)
76      c=0
77      for i in dis.keys():
78        if i==sito:
79          for a in dis[i]:
80            if c==0:
81              c=c+1
82              butt0=Gtk.RadioButton(label=a[0])
83              butt0.connect('toggled',self.on_button_toggled,a[1],sito)
84              box.pack_start(butt0,True,True,0)
85            else:
86              butt=Gtk.RadioButton.new_from_widget(butt0)
87              butt.set_label(a[0])
88              butt.connect('toggled',self.on_button_toggled,a[1],sito)
89              butt.set_active(False)
90              box.pack_start(butt,True,True,0)
91      return box
92
93    def disambigua(self,widget):
94      nodisambigua=False
95
96      if self.cercato:
97        self.ogg.Disambigua(self.q)
98        self.dis={}
99        if not self.q.empty():
100         self.dis=self.q.get()
101       self.cercato=False
102
103       if self.primavolta==False:
104         self.window.destroy()
105         self.boxcia.destroy()
106         self.boxwiki.destroy()
107         self.boxfree.destroy()
108       else:
109         self.primavolta=False
110       self.window=Gtk.Viewport()
111       self.boxcia=Gtk.Box(orientation=Gtk.Orientation.VERTICAL)
112       self.boxwiki=Gtk.Box(orientation=Gtk.Orientation.VERTICAL)
113       self.boxfree=Gtk.Box(orientation=Gtk.Orientation.VERTICAL)
```

```
114
115     if self.scrolledshow==False:
116
117       window2=self.builder.get_object('windowscelta')
118
119       if self.indicevisita==0:
120         URL=self.listadivisita[self.indicemax][1]
121       else:
122         URL=self.listadivisita[self.indicevisita -1][1]
123       self.window.destroy()
124       self.boxcia.destroy()
125       self.boxwiki.destroy()
126       self.boxfree.destroy()
127
128       self.window=Gtk.Viewport()
129
130       if URL.find('wolframalpha')!=-1:
131         nodisambigua=True
132       if URL.find('www.cia.gov')!=-1:
133         if 'cia' in self.dis:
134           if len(self.dis['cia'])<=1:
135             nodisambigua=True
136           else:
137             self.boxcia=self.crearadio(self.dis,'cia')
138             self.window.add(self.boxcia)
139         else:
140           nodisambigua=True
141       if URL.find('dbpedia.org')!=-1:
142         if 'dbpedia' in self.dis:
143           if len(self.dis['dbpedia'])<=1:
144             nodisambigua=True
145           else:
146             self.boxwiki=self.crearadio(self.dis,'dbpedia')
147             self.window.add(self.boxwiki)
148         else:
149           nodisambigua=True
150       if URL.find('freebase')!=-1:
151         if 'freebase' in self.dis:
152           if len(self.dis['freebase'])<=1:
153             nodisambigua=True
154           else:
155             self.boxfree=self.crearadio(self.dis,'freebase')
156             self.window.add(self.boxfree)
157         else:
158           nodisambigua=True
159
160       if nodisambigua:
161         self.builder.get_object('mexwarning').set_text('Nessuna␣
              disambiguazione␣disponibile')
```

```
162         self.builder.get_object('boxavvertimento').show()
163         return 0
164       else:
165         self.scrolledshow=True
166         window2.add(self.window)
167         window2.show_all()
168
169
170   def on_button_toggled(self,button,scelta,sito):
171     if button.get_active():
172       state='on'
173       if sito=='cia':
174         self.builder.get_object('Vai').set_uri('https://www.cia.gov/
              library/publications/the-world-factbook/geos/'+scelta+'.html')
175       if sito=='dbpedia':
176         self.builder.get_object('Vai').set_uri('http://dbpedia.org/
              resource/'+scelta)
177       if sito=='freebase':
178         self.builder.get_object('Vai').set_uri('http://www.freebase.com/'+
              scelta)
179       self.builder.get_object('Vai').show()
180     else:
181       state='off'
182
183   def Vai(self):
184     self.scrolledshow=True
185
186   def approfondisci(self,widget):
187     self.builder.get_object('boxavvertimento').hide()
188     self.builder.get_object('windowscelta').hide()
189     self.builder.get_object('Vai').hide()
190     self.scrolledshow=False
191     self.builder.get_object('WindowMain').resize(300,150)
192
193     if self.indicevisita==self.indicemax:
194       self.indicevisita=0
195     else:
196       self.indicevisita=self.indicevisita+1
197
198     url=self.listadivisita[self.indicevisita][1]
199     butt=self.builder.get_object('approfondisci')
200     butt.set_uri(url)
201     label='Approfondisci su \n'+self.listadivisita[self.indicevisita][0]
202     butt.set_label(label)
203
204   def translate(self,to_translate):
205     to_translate=to_translate.replace('\'',' ')
206     to_langage='en'
207     langage='it'
```

```
    agents = {'User-Agent':"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30)"}
    before_trans = 'class="t0">'
    link = "http://translate.google.com/m?hl=%s&sl=%s&q=%s" % (to_langage, langage, to_translate.replace(" ", "+"))
    request = urllib2.Request(link, headers=agents)
    page = urllib2.urlopen(request).read()
    result = page[page.find(before_trans)+len(before_trans):]
    result = result.split("<")[0]

    return result

  def cerca(self,widget):
    self.builder.get_object('boxavvertimento').hide()
    self.builder.get_object('windowscelta').hide()
    self.builder.get_object('Vai').hide()
    self.builder.get_object('WindowMain').resize(300,150)

    try:
      domanda=self.builder.get_object('entrydomanda').get_text()
      if domanda=='':
        raise ValueError
    except ValueError:
      self.builder.get_object('mexwarning').set_text('Devi prima inserire una domanda!')
      self.builder.get_object('boxavvertimento').show()
      return 0

    domanda=self.translate(domanda)

    window=Gtk.Window(title="Waiting...")
    window.set_position(Gtk.WindowPosition.CENTER_ON_PARENT)
    window.set_border_width(10)
    vbox = Gtk.Box(orientation=Gtk.Orientation.VERTICAL, spacing=6)
    window.add(vbox)
    label=Gtk.Label('Ricerca in corso...')
    progressbar = Gtk.ProgressBar()
    vbox.pack_start(label,True,True,0)
    vbox.pack_start(progressbar, True, True, 0)
    window.connect("delete-event", Gtk.main_quit)
    window.connect("destroy", lambda _: Gtk.main_quit())
    window.show_all()

    lista={}
    result=''
    q2=Queue.Queue()
    t=threading.Thread(target=self.ogg.Wolframalpha,args=(domanda,self.q,q2))
```

```
252         t.start()
253         t.join()
254         while not q2.empty():
255             result=q2.get()
256 
257         risultato=False
258         if result==''' or result==='(data not avaible)':
259             tokens= nltk.word_tokenize(domanda)
260         else:
261             tokens= nltk.word_tokenize(result)
262             risultato=True
263 
264         self.listadivisita=[]
265 
266         listathreads=[]
267         self.ogg.inizializza(self.q,tokens)
268 
269         t=threading.Thread(target=self.ogg.Ciagov)
270         t.start()
271         listathreads.append(t)
272 
273         t=threading.Thread(target=self.ogg.DBpedia)
274         t.start()
275         listathreads.append(t)
276 
277         t=threading.Thread(target=self.ogg.Freebase)
278         t.start()
279         listathreads.append(t)
280 
281         t1=Attesa(label,listathreads,progressbar,window)
282         t1.start()
283 
284         Gtk.main()
285 
286         for t in listathreads:
287             t.join()
288 
289         while not self.q.empty():
290             (link,URL)=self.q.get()
291             lista.update({link:URL})
292 
293         self.indicevisita=-1
294         self.indicemax=-1
295         self.scrolledshow=False
296         self.cercato=True
297 
298         if 'CiaGov' in lista and lista['CiaGov']!='':
299             self.listadivisita.append(('CiaGov',lista['CiaGov']))
300             self.indicevisita=self.indicevisita+1
```

```
301         self.builder.get_object('approfondisci').set_uri(self.listadivisita[
                self.indicevisita][1])
302         self.builder.get_object('boxapprofondisci').show()
303         self.builder.get_object('WindowMain').resize(300,150)
304
305       if 'DBpedia' in lista and lista['DBpedia']!='':
306         self.listadivisita.append(('DBpedia',lista['DBpedia']))
307         if self.indicevisita==-1:
308           self.builder.get_object('approfondisci').set_uri(self.
                listadivisita[self.indicevisita][1])
309           self.builder.get_object('boxapprofondisci').show()
310           self.builder.get_object('WindowMain').resize(300,150)
311         self.indicevisita=self.indicevisita+1
312
313       if 'Freebase' in lista and lista['Freebase']!='':
314         self.listadivisita.append(('Freebase',lista['Freebase']))
315         if self.indicevisita==-1:
316           self.builder.get_object('approfondisci').set_uri(self.
                listadivisita[self.indicevisita][1])
317           self.builder.get_object('boxapprofondisci').show()
318           self.builder.get_object('WindowMain').resize(300,150)
319         self.indicevisita=self.indicevisita+1
320
321       self.indicevisita=self.indicevisita+1
322       self.listadivisita.append(('Wolframalpha',lista['Wolframalpha']))
323       self.indicemax=self.indicevisita
324       self.indicevisita=0
325
326       butt=self.builder.get_object('approfondisci')
327       butt.set_uri(self.listadivisita[self.indicevisita][1])
328       label='Approfondisci su \n'+self.listadivisita[self.indicevisita][0]
329       butt.set_label(label)
330
331       del lista
332       if risultato==False and len(self.listadivisita)==0:
333         self.builder.get_object('mexwarning').set_text('Nessun risultato
                disponibile!')
334         self.builder.get_object('boxavvertimento').show()
335         return 0
```

A.3 Creazionedizionari.py

```
1 # -*- coding: utf-8 -*-
2 import sqlite3
3 import urllib,urllib2
4
5 class Dizionari:
6   def Dwiki(self):
```

```
7     Dwiki=open('DictWiki.txt','w')
8     with open('page_ids_en.ttl') as wiki:
9       for line in wiki:
10        if not line.startswith('#'):
11          parola=line.split(None,1)
12          parola=parola[0]
13          parola=parola.rstrip('>\n')
14          parola=parola.replace('<http://dbpedia.org/resource/','')
15          try:
16            b=unicode(parola)
17            Dwiki.write(b+'\n')
18          except:
19            pass
20    Dwiki.close()
21
22  def Dfree(self):
23    Dfree=open('DictFree.txt','w')
24    with open('freebase_links.nt') as free:
25      for line in free:
26        siti=line.split(None,3)
27        sito=siti[2]
28        sito=sito.lstrip('<http://rdf.freebase.com/')
29        sito=sito.rstrip('>')
30        sito=sito.replace('ns/','')
31        sito=sito.replace('.','/')
32        parola=siti[0].replace('<http://dbpedia.org/resource/','')
33        parola=parola.rstrip('>')
34        Dfree.write(parola+' '+sito+'\n')
35    Dfree.close()
36
37  def Dcia(self):
38    sock=urllib.urlopen('https://www.cia.gov/library/publications/the-
        world-factbook/geos/be.html')
39    htmlSource=sock.read()
40    sock.close()
41
42    stringa='<option value="">Please select a country to view</option>'
43    indice=htmlSource.find(stringa)
44    result=htmlSource[(indice+len(stringa)):]
45    indice=result.find('</select>')
46    result=result[:indice]
47
48    result=result.replace('<option value="../geos/','')
49    result=result.replace(' </option>','')
50    result=result.replace('\t','')
51    result=result.replace('\n\n','\n')
52    result=result.replace('.html">','')
53    result=result.lstrip('\n')
54
```

```
55      Dcia=open('DictCia.txt','w')
56      Dcia.write(result)
57      Dcia.close()
58
59    def sqlfree(self):
60      conn=sqlite3.connect('freebase.db')
61      c=conn.cursor()
62      c.execute('''CREATE TABLE FreeBase (indice, link)''')
63      valori=[] #lista tuple
64      with open("DictFree.txt") as f:
65        for x in f:
66          campi=x.split(None,1)
67          parola=campi[0]
68          parola=parola.replace(',',' ')
69          parola=parola.replace('_',' ')
70          parola=parola.replace('-',' ')
71          parola=parola.replace(':',' ')
72          parola=parola.replace('.',' ')
73          parola=parola.replace('%',' ')
74          parola=parola.replace('/',' ')
75          parola=parola.replace('\n','')
76          parola=parola.replace('  ',' ')
77          parola=parola.replace('( ','(')
78          parola=parola.replace(' )',')')
79          parola=parola.rstrip(' ')
80          sito=campi[1]
81          valori.append((parola,sito))
82      c.executemany('INSERT INTO FreeBase VALUES (?,?)',valori)
83      conn.commit()
84      c.close()
85      conn.close()
86
87    def sqlwiki(self):
88      conn=sqlite3.connect('dbpedia.db')
89      c=conn.cursor()
90      c.execute('''CREATE TABLE DBpedia (indice, link)''')
91      valori=[] #lista tuple
92      with open("DictWiki.txt") as f:
93        for x in f:
94          parola=x[:]
95          sito=x[:]
96          parola=parola.replace(',',' ')
97          parola=parola.replace('_',' ')
98          parola=parola.replace('-',' ')
99          parola=parola.replace('%',' ')
100         parola=parola.replace(':',' ')
101         parola=parola.replace('.',' ')
102         parola=parola.replace('/',' ')
103         parola=parola.replace('\n','')
```

```
104         parola=parola.replace('␣␣','␣')
105         parola=parola.replace('(␣','(')
106         parola=parola.replace('␣)',')')
107         valori.append((parola,sito))
108
109     c.executemany('INSERT␣INTO␣DBpedia␣VALUES␣(?,?)',valori)
110     conn.commit()
111     c.close()
112     conn.close()
113
114   def sqlcia(self):
115     conn=sqlite3.connect('ciagov.db')
116     c=conn.cursor()
117     c.execute('''CREATE TABLE CiaGov (indice, link)''')
118     valori=[] #lista tuple
119     with open("DictCia.txt") as f:
120       for x in f:
121         campi=x.split(None,1)
122         parola=campi[1]
123         sito=campi[0]
124         parola=parola.replace('\n','')
125         #li inserisco al contrario!
126         valori.append((parola+'␣',sito))
127
128     c.executemany('INSERT␣INTO␣CiaGov␣VALUES␣(?,?)',valori)
129     conn.commit()
130     c.close()
131     conn.close()
```

A.4 Sceltasito.py

```
1 # -*- coding: utf-8 -*-
2 import urllib,urllib2
3 from urllib2 import HTTPError
4 import nltk
5 from nltk.corpus import stopwords
6 import os.path
7 import types
8 import urllib
9 import webbrowser
10 import threading
11 import re
12 import sys
13 import time
14
15 import sqlite3
16
17 class sceltasito:
```

```
18  def inizializza(self,q,tokens):
19    self.__q_=q
20    self.__tokensnostop_=[]
21    self.tokens=[]
22
23    self.__Fdis_=[]
24    self.__Wdis_=[]
25    self.__Cdis_=[]
26    self.wnl=nltk.WordNetLemmatizer()
27
28    for t in tokens:
29      t=t.lower()
30      if (not t in stopwords.words('english') and len(t)>1) or t=='|':
31        self.tokens.append(self.wnl.lemmatize(t))
32
33    #nel caso di risposte multiple
34    supporto=[]
35    if '|' in self.tokens:
36      appoggio=[]
37      for i in self.tokens:
38        if i!='|':
39          appoggio.append(i)
40        else:
41          supporto.append(appoggio)
42          appoggio=[]
43      supporto.append(appoggio)
44
45      if len(supporto)>2:
46        for i in range(0,2):
47          self.__tokensnostop_.append(supporto[i])
48      else:
49        for i in range(len(supporto)):
50          self.__tokensnostop_.append(supporto[i])
51    else:
52      self.__tokensnostop_.append(self.tokens)
53
54    if self.__tokensnostop_==[]:
55      self.__tokensnostop_=tokens
56
57  def Disambigua(self,q):
58    tuples=[]
59    dis={}
60    if len(self.__Fdis_)>1:
61      for i in self.__Fdis_:
62        tuples.append(i)
63      dis.update({'freebase':tuples})
64
65    tuples=[]
66    if len(self.__Wdis_)>1:
```

```
67        for i in self.__Wdis_:
68          tuples.append((i[0],i[1]))
69        dis.update({'dbpedia':tuples})
70
71      tuples=[]
72      if len(self.__Cdis_)>1:
73        dis.update({'cia':self.__Cdis_})
74
75      q.put(dis)
76
77    def Freebase(self):
78      try:
79        conn = sqlite3.connect('freebase.db')
80        c = conn.cursor()
81      except IOError:
82        pass
83
84      ris=[]
85      probabile=[]
86      probabile2=[]
87      Fdic=[]
88
89      for i in self.__tokensnostop_:
90        n=len(i)
91        domanda='SELECT * FROM FreeBase WHERE indice LIKE ?'+' AND indice
            LIKE ?'*(n-1)
92        dati=[]
93        for h in i:
94          dati.append('%'+h.capitalize()+'%')
95        c.execute(domanda,dati)
96        probabile.append(c.fetchall())
97      c.close()
98      conn.close()
99
100     for i in probabile:
101       for x in i:
102         probabile2.append(x)
103     del probabile
104
105     URLFree=''
106     listatokens=[]
107     supporto=[]
108
109     for i in self.__tokensnostop_:
110       if len(i) ==1:
111         for x in probabile2:
112           if len(x[0])==len(i[0]):
113             self.__Fdis_.append((x[0],x[1]))
```

```
114            if x[0].startswith(i[0].capitalize()+'␣(') or x[0].startswith('
                 The␣'+i[0].capitalize()+'␣(') :
115              supporto.append((x[0],x[1]))
116          if self.__Fdis_!=[]:
117            URLFree='http://www.freebase.com/'+self.__Fdis_[0][1]
118        else:
119          listatokens.append(i)
120
121      for x in probabile2:
122        if x[0].find('File')!=-1 or x[0].find('Category')!=-1 or x[0].find('
             Template')!=-1:
123          pass
124        else:
125          for t in listatokens:
126            cont=0
127            contmax=len(t)
128            for i in t:
129              l=len(t)
130              if x[0].find(i.capitalize()+'␣')!=-1 or x[0].find(i+')')!=-1
                   or x[0].find('('+i)!=-1 or x[0].find(i.capitalize()+')')
                   !=-1 or x[0].find('('+i.capitalize())!=-1:
131                cont=cont+1
132              else:
133                lista=x[0].split("␣")
134                for t in lista:
135                  if len(t)==l:
136                    cont=cont+1
137            if cont==contmax:
138              Fdic.append((x[0],x[1]))
139
140      del probabile2
141
142      for a in listatokens:
143        nparole=len(a)
144        for i in Fdic:
145          lista=i[0].split("␣")
146          nparole2=len(lista)
147          if i[0].find('(')!=-1:
148            inizio=i[0].find('(')
149            parentesi=i[0][inizio:]
150            listaparentesi=parentesi.split("␣")
151            conto=0
152            for parole in a:
153              if parentesi.find(parole)!=-1 or parentesi.find(parole.
                   capitalize())!=-1 :
154                conto=conto+1
155            nparole2=nparole2-len(listaparentesi)+conto
156            senzaparentesi=i[0][:inizio]
157            lista=senzaparentesi.split("␣")
```

```
158
159         for t in lista:
160           if not t in stopwords.words('english'):
161             pass
162           else:
163             nparole2=nparole2-1
164         if nparole2==nparole:
165           supporto.append((i[0],i[1]))
166
167     del Fdic
168
169     tupla=()
170     lung=sys.maxint
171     for i in supporto:
172       lungI=len(i[0])
173       if lungI<lung:
174         lung=lungI
175         tupla=(i)
176
177     if tupla!=():
178       self.__Fdis_.append(tupla)
179       for i in supporto:
180         if i!=tupla:
181           self.__Fdis_.append(i)
182
183     if URLFree==''' and self.__Fdis_!=[]:
184       URLFree='http://www.freebase.com/'+self.__Fdis_[0][1]
185     print URLFree
186
187     if URLFree!='':
188       self.__q_.put(('Freebase',URLFree))
189
190   def DBpedia(self):
191     try:
192       conn = sqlite3.connect('dbpedia.db')
193       c = conn.cursor()
194     except IOError:
195       pass
196
197     ris=[]
198     probabile=[]
199     probabile2=[]
200     Wdic=[]
201
202     for i in self.__tokensnostop_:
203       n=len(i)
204       domanda='SELECT␣*␣FROM␣DBpedia␣WHERE␣indice␣LIKE␣?'+'␣AND␣indice␣
            LIKE␣?'*(n-1)
205       dati=[]
```

```
206         for h in i:
207           dati.append('%'+h.capitalize()+'%')
208         c.execute(domanda,dati)
209         probabile.append(c.fetchall())
210       c.close()
211       conn.close()
212
213       for i in probabile:
214         for x in i:
215           probabile2.append(x)
216
217       del probabile
218
219       URLWiki=''
220       listatokens=[]
221       supporto=[]
222
223       for i in self.__tokensnostop_:
224         if len(i) ==1:
225           for x in probabile2:
226             if len(x[0])==len(i[0]):
227               self.__Wdis_.append((x[0],x[1]))
228             if x[0].startswith(i[0].capitalize()+'␣(') or x[0].startswith('
                  The␣'+i[0].capitalize()+'␣(') :
229               supporto.append((x[0],x[1]))
230
231           if self.__Wdis_!=[]:
232             URLWiki='http://dbpedia.org/resource/'+self.__Wdis_[0][1]
233         else:
234           listatokens.append(i)
235
236       for x in probabile2:
237         if x[0].find('File')!=-1 or x[0].find('Category')!=-1 or x[0].find('
              Template')!=-1:
238           pass
239         else:
240           for t in listatokens:
241             cont=0
242             contmax=len(t)
243
244             for i in t:
245               l=len(i)
246
247               if x[0].find(i.capitalize()+'␣')!=-1 or x[0].find(i+')')!=-1
                    or x[0].find('('+i)!=-1 or x[0].find(i.capitalize()+')')
                    !=-1 or x[0].find('('+i.capitalize())!=-1:
248                 cont=cont+1
249               else:
250                 lista=x[0].split("␣")
```

```
251                 for t in lista:
252                   if len(t)==l:
253                     cont=cont+1
254
255             if cont==contmax:
256               Wdic.append((x[0],x[1]))
257
258     del probabile2
259
260     for a in listatokens:
261       nparole=len(a)
262       for i in Wdic:
263         lista=i[0].split(" ")
264         nparole2=len(lista)
265
266         if i[0].find('(')!=-1:
267           inizio=i[0].find('(')
268           parentesi=i[0][inizio:]
269           listaparentesi=parentesi.split(" ")
270           conto=0
271           for parole in a:
272             if parentesi.find(parole)!=-1 or parentesi.find(parole.
                  capitalize())!=-1 :
273               conto=conto+1
274           nparole2=nparole2-len(listaparentesi)+conto
275           senzaparentesi=i[0][:inizio]
276           lista=senzaparentesi.split(" ")
277
278         for t in lista:
279           if not t in stopwords.words('english'):
280             pass
281           else:
282             nparole2=nparole2-1
283
284         if nparole2==nparole:
285           supporto.append((i[0],i[1]))
286
287     del Wdic
288
289     lung=sys.maxint
290     tupla=()
291     for i in supporto:
292       lungI=len(i[0])
293       if lungI<lung:
294         lung=lungI
295         tupla=i
296     if tupla!=():
297       self.__Wdis_.append(tupla)
298       for i in supporto:
```

```
299         if i!=tupla:
300           self.__Wdis_.append(i)
301
302     if URLWiki==''  and  self.__Wdis_!=[]:
303       URLWiki='http://dbpedia.org/resource/'+self.__Wdis_[0][1]
304     print URLWiki
305     if URLWiki!='':
306       self.__q_.put(('DBpedia',URLWiki))
307
308
309   def Ciagov(self):
310     try:
311       conn = sqlite3.connect('ciagov.db')
312       c = conn.cursor()
313     except IOError:
314       pass
315     ris=[]
316     for t in self.__tokensnostop_:
317       for i in t:
318         c.execute('SELECT␣*␣FROM␣CiaGov␣WHERE␣indice␣LIKE␣?␣OR␣link␣LIKE␣?
              ',['%'+i.capitalize()+'%',i])
319         ris=c.fetchall()
320     c.close()
321     conn.close()
322     if ris!=[]:
323       for x in ris:
324         trovato=False
325         for t in self.__tokensnostop_:
326           for i in t:
327             if x[0].find(i.capitalize()+'␣')!=-1 or x[0].find(i.capitalize
                  ()+',')!=-1 or x[1].find(i)!=-1:
328               trovato=True
329           if trovato:
330             self.__Cdis_.append(x)
331       ciagovURL=''
332       if self.__Cdis_!=[]:
333         ciagovURL='https://www.cia.gov/library/publications/the-world-
              factbook/geos/'+self.__Cdis_[0][1]+'.html'
334       if ciagovURL!='':
335         self.__q_.put(('CiaGov',ciagovURL))
336
337   def Wolframalpha(self,frase,q,q2):
338     URL='http://www.wolframalpha.com/input/?i='+frase
339     q.put(('Wolframalpha',URL))
340     sock=urllib.urlopen(URL)
341     htmlSource=sock.read()
342     sock.close()
343     if 'know␣how␣to␣interpret␣your␣input' in htmlSource or 'Using␣closest␣
          Wolfram|Alpha␣interpretation:' in htmlSource :
```

```
344         return 0
345     if 'Result<span class="colon">:</span></h2>' in htmlSource:
346         webbrowser.open(URL)
347 
348         f=open('wolf.html','w')
349         f.write(htmlSource)
350         f.close()
351         indice=htmlSource.find('context.jsonArray.popups.pod_0200.push( {"
            stringified": ')
352         result=htmlSource[indice:]
353         indice=result.find('} catch(e) { }')
354         result=result[:indice]
355         indice=result.find(': "')
356         indice=indice+3
357         result=result[indice:]
358         indice=result.find('"')
359         result=result[:indice]
360         #gestione lettere
361         result=result.replace('&iacute;','i')
362         result=result.replace('\\\'s','')
363 
364         if '\\' in result:
365             indice=result.find('\\')
366             result=result[:indice]
367 
368         if '(' in result:
369             indice=result.find('(')
370             result=result[:indice]
371 
372         result=result.replace('&#39;','\'')
373         result=result.replace('&quot','')
374 
375         risultatocapitale=''
376         if frase.find('capital')!=-1:
377             lista=result.split(',')
378             if len(lista)>=3:
379                 risultatocapitale=str(lista[0])+','+str(lista[-1])
380 
381         if risultatocapitale!='':
382             q2.put(risultatocapitale)
383         else:
384             q2.put(result)
385     else:
386         webbrowser.open(URL)
```

Bibliografia

1. Wikipedia Foundation.
   http://www.wikipedia.org.
2. Dizionario Treccani.
   http://www.treccani.it/.
3. RDF per la rappresentazione della conoscenza.
   http://www.w3c.it/papers/RDF.pdf.
4. A Semantic Web Primer for Object-Oriented Software Developers.
   http://www.w3.org/TR/sw-oosd-primer/.
5. SQLite.
   http://www.sqlite.org/index.html.
6. SQLite3 for Python.
   http://docs.python.org/2/library/sqlite3.html.
7. Python.
   http://www.python.org/.
8. Python.
   http://docs.python.org/2/.
9. Python. Drive Into Python.
   http://it.diveintopython.net/toc/.
10. Wolframalpha.
    http://www.wolframalpha.com/about.html.
11. DBpedia.
    http://dbpedia.org/About.
12. Freebase.
    http://www.freebase.com/.

13. CIA.
https://www.cia.gov/library/publications/the-world-factbook/.

14. NLTK
http://www.nltk.org/.

15. NLTK, XML.
http://www.isgroup.unimore.it/corsi/gavi/materiale-didattico-2013/.