UNIVERSITÀ DI PISA

Corso di Laurea in Informatica Umanistica

RELAZIONE

Il Web Semantico e le Digital Humanities

Una applicazione per la storia digitale

Candidato: *Giacomo Corsini*

Relatore: *Prof. Alessandro Lenci*

Correlatore: *Prof.ssa Maria Simi*

Anno Accademico 2014-2015

I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A "Semantic Web", which makes this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The "intelligent agents" people have touted for ages will finally materialize.

(Tim Berners Lee. Weaving the Web, 1999)

Indice

1. Introduzione: Cos'è il Web semantico

1.1 Un approccio generale

Nella società dell'informazione in cui viviamo siamo sempre più abituati ad utilizzare, e spesso non riusciamo a fare a meno, della rete informatica più estesa al mondo, ovvero Internet, e in particolare di uno dei suoi servizi più recenti: il Web, nato nel 1989.
Dopo i primi timidi passi al Cern di Ginevra, il Web si è evoluto in un media vero e proprio a cui l'utente può accedere attingendo ad una grande quantità di informazioni fruendone però sostanzialmente come i Media tradizionali, senza una possibile interazione.
Negli ultimi anni la Rete è diventata uno strumento che permette al singolo utente di interagire con una determinata applicazione Web: egli diventa allo stesso tempo fruitore e produttore di contenuti, grazie allo sviluppo dei blog, dei social network, della possibilità di avere sempre più contenuti multimediali.
Quest'ultima versione, denominata Web 2.0[1] (in contrapposizione al Web 1.0, descritto all'inizio), ha costituito senza dubbio una grossa evoluzione tecnologica, ma anche socio-culturale, che ha rivoluzionato il modo di produrre e di ricercare informazione ed ha incrementato il numero degli utenti che navigano quotidianamente in Rete.
In sintesi siamo in presenza di un potente strumento pensato più per le persone che non per le macchine e proprio questo aspetto si rivela il suo limite principale. In altre parole, il Web attuale è ***machine-readable*** e non ***machine-understandable***.
Quando su un motore di ricerca viene digitata una "query string", l'algoritmo cerca di trovare delle corrispondenze di quella particolare sequenza di lettere confrontandola con i termini che il motore di ricerca ha in elenco ma dei quali ignora il significato.
Il computer, al livello attuale, interpreta tutte le informazioni come pure sequenze di bit prive di significato; in altre parole non tiene traccia dell'aspetto *semantico*.
In generale l'aggettivo "semantico" indica quella parte della linguistica che studia il

[1] http://it.wikipedia.org/wiki/Web_2.0

rapporto tra *significante* (ossia l'immagine acustica o visiva, l'elemento formale del segno) e la *realtà extralinguistica* (l'ambito, l'ente, l'oggetto) a cui si riferisce[2].
Tim Berners Lee, l'ideatore del WWW, in un articolo scritto nel settembre 1998[3], dice testualmente:

Il Web fu progettato come uno spazio di informazioni, con l'obiettivo di essere utile non solo per la comunicazione uomo-uomo, ma affinché anche le macchine potessero avere la possibilità di partecipare e dare il loro contributo. Uno dei maggiori ostacoli è stato il fatto che la maggior parte dell'informazione sul Web è progettata per essere fruita dall'uomo, […] la struttura dei dati non è riconoscibile per un robot che naviga il Web. […] l'approccio del Web Semantico, invece, sviluppa linguaggi per esprimere le informazioni in una forma accessibile e processabile da una macchina.

In generale il Web è nato e si presenta tutt'oggi come un insieme di documenti, collegati in modo univoco tra di loro, mediante URI di due tipi:

- *Collegamenti sintattici* legati al funzionamento di codice di programmazione: sono altamente affidabili, in quanto un URI punta ad una risorsa in modo univoco
- *Collegamenti semantici*, ovvero che descrivano la risorsa verso cui puntano: sono ancora poco sviluppati

Gli utenti si orientano sul Web grazie all'esperienza di navigazione e alla capacità di evocazione che possono avere parole o espressioni chiave; in definitiva queste sono peculiarità tipiche degli esseri umani, che le macchine non sono ancora in grado di raggiungere.
Il primo a parlare di ***Web Semantico*** **(**o ***Web 3.0*****)** è l'ideatore stesso del WWW, Tim Berners Lee, che nel 2001[4] propone lo sviluppo di agenti intelligenti in grado di:

- comprendere il significato dell'informazione

[2] http://www.treccani.it/enciclopedia/significante/
[3] Berners Lee T., *Semantic Web Road map. An attempt to give a high-level plan of the architecture of the Semantic WWW*. Settembre 1998, http://www.w3.org/DesignIssues/Semantic.html
[4] Berners-Lee T., Hendler J., Lassila O., (2001), "The Semantic Web", in *Scientific American*, http://www.cs.umd.edu/~golbeck/LBSC690/SemanticWeb.html

- creare percorsi compilati attraverso le informazioni richieste dall'utente, guidandolo verso di esse.
- Muoversi autonomamente attraverso la rete collegando logicamente le informazioni pertinenti con l'argomento richiesto dall'utente

Lo stesso Berners Lee in un articolo su Scientific American del maggio 2001[5], scrive:

"Il Web Semantico è un'estensione del Web corrente in cui le informazioni hanno un ben preciso significato e in cui computer e utenti lavorano in cooperazione".

Esso ha quindi la funzione di permettere l'interscambio di informazione tra uomo e macchine, rendendo queste ultime in grado di accedere ai dati dal punto di vista semantico e quindi di restituire risultati più accurati.
Per conferire significato ai dati, la soluzione è quella di utilizzare *schemi* per associare archivi di informazioni; essi non sono altro che insiemi di regole che stabiliscono come debbano essere organizzati i dati (vedi ad es. gli XML schema) e le relazioni tra di loro.
I dati devono a loro volta essere associati a classi dello schema, che definiscono il contesto e le caratteristiche della risorsa, tramite i cosiddetti *metadati* (dal greco μετὰ "oltre, dopo" e dal latino *datum* "informazione", dunque "dati su altri dati"), che forniscono informazioni sulla risorsa stessa in un linguaggio comprensibile alle macchine. In tal modo è possibile descrivere e rendere automatici i collegamenti tra dati.
Ritornando all'esempio iniziale di query ad un motore di ricerca i metadati, per usare una definizione di Berners Lee, "*forniscono proprietà semantiche alle risorse Web dandone informazioni comprensibili ad un computer*".[6]
Le conseguenze di quest'ultima affermazione sono molteplici e non si limitano ad una migliore qualità in termini di ricerca sul Web, ma renderebbero le macchine capaci di un certo grado di ragionamento oltre a permettere un certo grado di condivisione delle informazioni ed interoperabilità tra applicazioni.

[5] Berners-Lee T., Hendler J., Lassila O., *Op. cit.*

[6] Berners-Lee T, Connolly D., Swick R. R., (1999), *Web Architecture: Describing and Exchanging* Data, http://www.w3.org/1999/04/WebData

Il campo della ricerca di informazioni sul Web non è l'unico in cui metadati giocano un ruolo chiave; sono infatti lo strumento che permette di rendere fruibile nel tempo una risorsa generica (non necessariamente presente sul Web), tenendo traccia del suo autore, del formato e degli strumenti utilizzati per elaborarla, ecc. (vedi più in dettaglio §3.3). Si parla in questo caso di *metadati conservativi*.

In sintesi quella del Web Semantico è una struttura composta essenzialmente da 3 livelli:

- *Dati*
- *Metadati*, tramite cui i dati sono mappati in uno schema
- *Schemi*, in cui si esplicitano relazioni fra concetti, ovvero tra *classi* di dati

Esso ha la seguente architettura "piramidale":

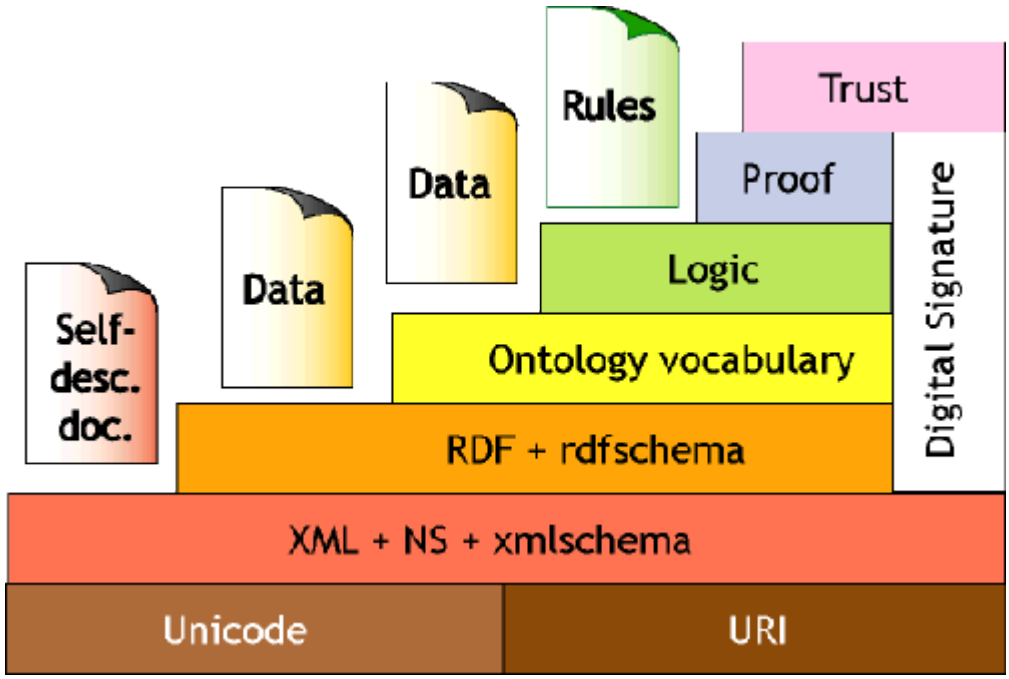

Figura n. 1

Il *layer-cake* del Web Semantico[7]

[7] http://www.w3.org/2000/Talks/1206-xml2k-tbl/Overview.html

1. Alla base della piramide troviamo due standard:
   - UNICODE per la codifica univoca di tutti i caratteri esistenti
   - URI per l'identificazione precisa di risorse su Internet
2. Subito al di sopra troviamo XML (*eXtensible Markup Language*) con i NameSpace e gli XMLSchema
3. XML consente di dare informazioni sulla semantica degli oggetti trattati ma presenta dei limiti. Per questo motivo è stato creato un nuovo linguaggio standard detto RDF (*Resource Description Framework*) che deriva da XML. RDF permette di definire la semantica dei tag XML e dunque è lo strumento che consente l'interazione tra applicazioni "machine-understandable". In sostanza abbiamo un modello per descrivere le risorse composto da triple della forma: Risorsa, proprietà, valore.
4. Salendo ancora nella piramide abbiamo il *livello ontologico*: le ontologie e i linguaggi che le descrivono (come OWL, *Web Ontology Language*), permettono di definire la semantica dei dati e le relazioni tra di essi espandendo gli RDF schema che possono contenere delle limitazioni. Possiamo notare come la *firma digitale* sia un elemento fondamentale per gran parte dei livelli superiori: la certificazione dell'integrità e dell'autenticità dei dati è di grande importanza anche nel Web attuale.
5. Il *livello logico* è la sfida per il futuro, in quanto è una parte del *layer cake* che ancora non è stata sviluppata. L'obiettivo è quello di rendere le macchine capaci di ragionare per inferenza sulla base di principi logici predeterminati. In sostanza le asserzioni presenti nelle ontologie possono essere utilizzate per ricavare nuova conoscenza.

1.2 Obiettivi del progetto

Partendo da queste considerazioni e dopo aver studiato lo stato dell'arte del Web Semantico, ci si è prefissi di realizzarne un prototipo in scala ridotta partendo da un corpus "statico", privo di qualsiasi tipo di collegamento ad altre risorse.
Il corpus in oggetto è composto da bollettini delle due Guerre Mondiali, accuratamente selezionati, digitalizzati e successivamente elaborati per mezzo degli strumenti con cui opera la Linguistica Computazionale.
Il corpus così digitalizzato è stato arricchito con tutta una serie di collegamenti

ipertestuali ed altri dati che permetteranno di conferire ad esso proprietà semantiche. Esso racchiude infatti una notevole quantità di informazioni che in prima battuta risultano "nascoste". Il nostro compito è quindi quello di giungere ad un corpus strutturato che consenta di accedere a tutti i dati più significativi senza ambiguità, rilasciandoli poi come "Linked Open Data" per nuove elaborazioni.
In sostanza possiamo parlare di un progetto che, nel pieno spirito delle Digital Humanities, riesce a coniugare la ricerca storica con la Linguistica Computazionale, creando un prototipo del nuovo Web e mostrandone tutte le sue potenzialità.

2. Linked data

2.1 Introduzione

In prima approssimazione, i Linked Data indicano quelle metodologie che consentono di pubblicare, condividere e connettere dati per mezzo di URI che li identifichino univocamente e che permettano, tramite tag ed etichette adatte, di associare loro una rappresentazione "Machine-Readable".
Si viene pertanto a costituire un vero e proprio reticolo di dati collegati fra di loro, appartenenti ad uno specifico dominio e connessi ad altri domini in una rete sempre più fitta ed ampia.
La tecnologia dei Linked Data deriva dalla ricerca condotta sul Web Semantico e ha dato origine al cosiddetto *Web dei Dati*, definito come"*Web of things in the world, described by data on the Web*"[8]; un Web di oggetti del mondo, descritto da dati presenti sul Web stesso.
Il Web tradizionale (ipertestuale) ha quindi scarse connessioni tra dati ed è assimilabile ad un contenitore di dati non relazionati, fruibili sostanzialmente solo dall'uomo.
Il Web di dati (Web Semantico) è costituito invece da oggetti relazionati tra di loro: la stessa descrizione del dato costituisce a sua volta un oggetto Web ed è riutilizzabile. Le entità descritte sono altamente strutturate e, una volta definite,

[8] Bizer, C., Heath, T. and Berners-Lee, T. (2009). "Linked Data - the story so far", *International Journal on Semantic Web and Information Systems*, 5, (3), 1-22

diventano fruibili principalmente dalle macchine, più che dall'uomo.
Si viene pertanto a creare un insieme di singoli dati, in sé autonomi, che possono essere messi in collegamento in modi diversi ed utilizzati da applicazioni differenti.
In particolare è possibile fare un parallelo tra Database relazionali e Web Semantico: un record è assimilabile ad un nodo RDF, il nome di un campo del DB (colonna) ad una proprietà RDF, mentre il campo stesso del DB è riconducibile al valore della proprietà RDF.
In Rete esistono diversi siti in cui si possono condividere e scaricare liberamente Dataset Open. In Italia possiamo contare su un portale appositamente creato allo scopo: DatiOpen, raggiungibile all'indirizzo http://www.datiopen.it/it.
Da segnalare inoltre Datahub, piattaforma per il trattamento di dati creata dall'Open Knowledge Foundation: http://datahub.io/it/.
Per avere un'idea dello sviluppo dei LD è sufficiente visualizzare il *Linked Open Data cloud* aggiornato all'agosto 2014:

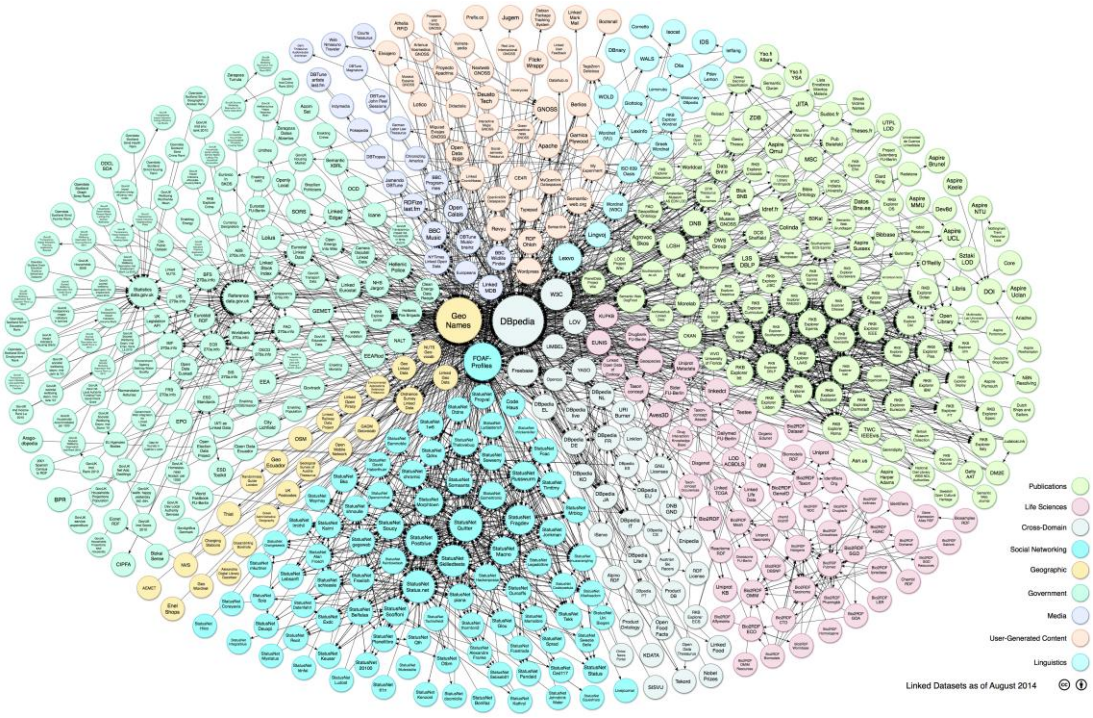

Figura n. 2 Linking Open Data cloud diagram 2014, di Max Schmachtenberg, Christian Bizer, Anja Jentzsch e Richard Cyganiak. http://lod-cloud.net/

L'immagine mostra tutti i dataset open pubblicati sulla Rete come Linked Data dai collaboratori della LOD community project ed è in continua espansione.

2.2 URI

Gli URI (Uniform Resource Identifier) sono un sistema di identificatori che permette di individuare le risorse in modo univoco sul Web. Per questa ragione essi hanno un ruolo chiave nel Web semantico: gli URI sono utilizzati da RDF per tenere traccia dell'informazione in un documento. Nelle triple RDF, soggetto, oggetto e predicato sono identificati ciascuno da un URI. In tal modo si tiene traccia di ogni informazione, ricercandola e recuperandola senza ambiguità.
Sul Web i più comuni URI sono gli URL (Uniform Resource Locator) che, oltre a identificare una risorsa, la localizzano; e poiché la pagina Web descrive un oggetto, l'URL della pagina è a sua volta l'URI di tale oggetto.
Gli URI non vengono controllati da un sistema centralizzato, ma possono essere creati da chiunque; pertanto è possibile avere URI diversi che puntano alla stessa risorsa, senza poterne tenere traccia in qualche modo. Per questo motivo l'URI rappresenta uno strumento potente ma da solo insufficiente a determinare il significato (la semantica) di una risorsa. Approfondire differenze URI/URL

2.3 RDF

I linked data vengono espressi tramite lo standard RDF (Resource Description Framework) che è alla base del Web semantico. Esso non è un formato di dati ma un *data model*, ossia un *formalismo* per la rappresentazione, la codifica, lo scambio e il riutilizzo di metadati strutturati. In quanto modello astratto, esso può essere implementato in vari formati (RDF/XML, N3, NTriple, Json ecc.). Permette dunque l'interscambio di dati sul Web e si basa sulla *logica dei predicati*. Secondo quest'ultima, le informazioni sono esprimibili mediante *asserzioni* costituite da **triple**, ovvero proposizioni elementari costituite da:

soggetto, predicato, oggetto

Vediamo con uno schema:

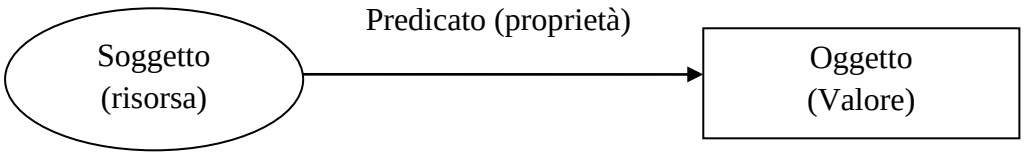

Figura n. 3

Le triple permettono di dire che un'entità (soggetto) ha delle proprietà (predicato) con certi valori (oggetto). Esse possono essere espresse in vari formati; **RDF/XML** è quello proposto dal W3C ed è quindi uno dei più comuni.
XML (*eXtensible Markup Language*) è un linguaggio che permette di definire altri linguaggi di markup, dunque è un *metalinguaggio* di markup; nello specifico si presenta come un insieme di regole standard (dette *specifiche*[9]) per modellare la struttura di documenti e di dati.
Essendo standard, le specifiche di XML lo rendono indipendente dalla piattaforma o da uno specifico produttore; tuttavia, pur consentendo la definizione della struttura di documenti e di dati, non ne consente la dichiarazione del tipo o la loro presentazione. Per colmare questo limite di XML è stato sviluppato RDF, che presenta il vantaggio di poter definire il tipo e le proprietà degli oggetti conferendo ai tag XML proprietà semantiche. Una descrizione RDF è un documento XML che contiene sia tag predefiniti, sia tag "liberi" per la descrizione della risorsa.
In sostanza RDF modifica XML rendendolo capace di descrivere, tramite i *metadati*, un'entità insieme alle sue proprietà e alle relazioni con altri oggetti.
Tale modello ha la caratteristica di rappresentare tutti gli elementi della tripla (soggetto, predicato, oggetto) tramite URI (Uniform Resource Identifier) che puntano alle rispettive risorse.
Un tipo di URI molto utilizzato anche nei Linked Data sono gli ***URL*** (Uniform Resource Locator) che in questo caso contengono la descrizione della risorsa in formato RDF.
Un esempio di Linked Data con un URI che fa riferimento ad una risorsa esterna è presente nello schema seguente:

[9] Specifiche XML: http://www.w3.org/XML/

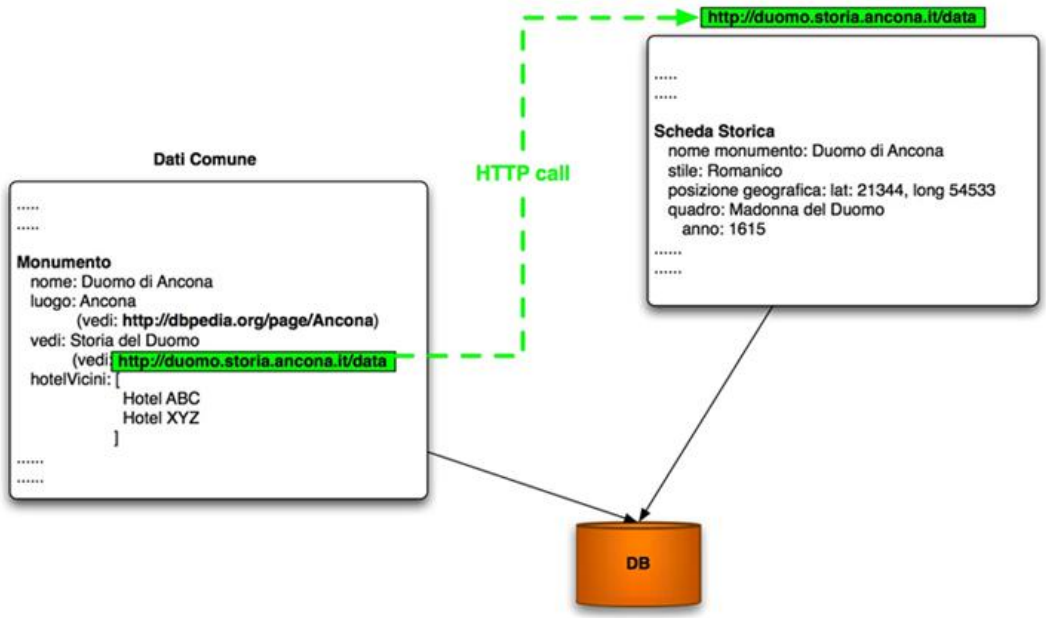

Figura n. 4 (da http://www.linkedopendata.it)

Lo schema è un esempio particolarmente efficace di Linked Data: in questo caso si parte dalla scheda del Duomo della città di Ancora. È possibile notare che le voci (nome, luogo ecc.), assimilabili ai campi di un Database relazionale, puntano a loro volta ad altri dati *collegati* - mediante URI - all'entità di partenza (il Duomo). È possibile per esempio risalire ai dati relativi alla città in cui si trova il monumento, all'elenco di locali nelle vicinanze o ancora alla sua storia, accedendo così ad un altro set di dati.

In sostanza ogni dato risulta collegato a tutti gli altri in una vera e propria rete ordinabile secondo gerarchie.

Tuttavia RDF necessita di ulteriori tecnologie per rendere possibile l'interoperabilità tra applicazioni. Ad esempio, due sistemi potrebbero far uso di URI diverse per riferirsi alla stessa risorsa, pertanto le rispettive asserzioni su quello stesso elemento risulterebbero non confrontabili.

Vediamo quali strumenti esistono per ovviare a questo problema.

RDF è costituito da due componenti:

- RDF *Model and Syntax: descrive il modello di dati RDF e l'XML relativo*
  Ogni oggetto è visto come un tripla (soggetto, predicato, oggetto).

- RDF *Schema: per la definizione di vocabolari per i metadati e relazioni tra entità*

Vediamoli nel dettaglio.

2.4 RDF Data Model

Per RDF ogni risorsa è identificabile in modo univoco con un URI, ed è descritta da una tripla contenente:

Soggetto: la *risorsa*, che può essere una pagina Web, un'immagine, un file ecc.

Predicato: la *proprietà* della risorsa

Oggetto: il *valore* della proprietà; descrive le caratteristiche di quella risorsa

Vediamo un esempio semplice di tripla:

Soggetto: http://www.nomesito.it/Libri/IPromessiSposi.html
Predicato: http://www.nomesito.it/hasAuthor
Oggetto: http://www.nomesito.it/Autori/Alessandro Manzoni

Che graficamente sarà rappresentata da nodi e archi:

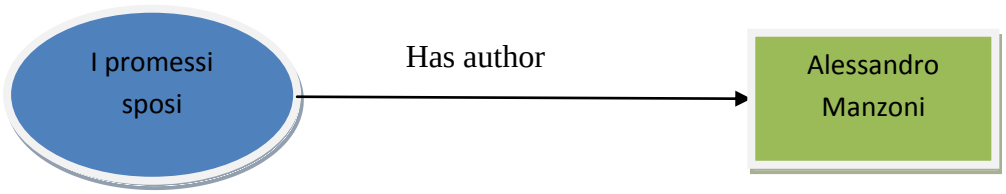

Figura n. 5

E avrà il seguente schema XML:

```
<rdf:Description
rdf:about="http://www.host.org/I_promessi_sposi">
   <s:Autore>http://www.host.org/Autori/Alessandro_Manzoni
   </s:Autore>
</rdf:Description>
```

Tali asserzioni, espresse da triple, possono essere a loro volta riutilizzate per generare nuova informazione generando una rete semantica sempre più fitta.

2.5 RDFS - RDF Schema

Con RDFS[10] è possibile definire vocabolari che specifichino classi di risorse e proprietà che le definiscano; quindi il significato, le caratteristiche e le relazioni di un insieme di risorse.
Lo scopo di RDFS è quello di validare un valore o restringere il dominio di applicazione di una proprietà; esso non vincola la struttura del documento come lo Schema XML o la DTD, ma fornisce specifiche per l'interpretazione del documento stesso. Lo Schema è definito per mezzo della sintassi stessa di RDF.
RDFS può quindi dare una *struttura semantica* ad un particolare dominio di dati, organizzandolo secondo una gerarchia di concetti e relazioni suddivisa in *classi*, *sottoclassi* e loro *attributi*, ovvero le loro proprietà.

Vediamo un esempio di RDF Schema:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:a="http://www.nomesito.it/data/nomeontologia#">
<rdfs:Class rdf:about
resource="http://www.nomesito.it/data/nomeontologia#Corso"/>
  <rdfs:subClassof rdf:resource="#Resource"/>
</rdfs:Class>
<rdfs:Property rdf:about
 resource="http://www.nomesito.it/data/nomeontologia#Titolo"/>
  <rdfs:domain rdf:resource="#Corso"/>
  <rdfs:range rdf:resource="#Literal"/>
</rdfs:Class>
```

All'inizio abbiamo il *namespace* e l'URI dell'*Ontologia* (uno o più schemi a cui sono riferiti i vari elementi).
Seguono poi i dati veri e propri. La classe "Corso" è dichiarata come sottoclasse di una generica classe Resource ed ha una proprietà (*property*) chiamata"Titolo".
Come dominio (*domain*) la proprietà "Titolo" ha la classe "Corso" ed appartiene al *range* Literal che in RDF identifica i testi.

[10] http://www.w3.org/TR/rdf-schema/

In generale, RDFS può essere utile per la descrizione di diverse entità nella Rete, come ad esempio[11]:

- descrizione del contenuto di un sito Web o di una biblioteca digitale
- implementazione di *agenti software intelligenti*
- classificazione del contenuto di una risorsa
- descrizione di un insieme di pagine
- determinazione dei criteri di proprietà intellettuale delle singole pagine
- criteri di *privacy preference* degli utenti e le *privacy policies* di un sito Web
- con il meccanismo della firma digitale, contribuire alla creazione del *Web of Trust*, per le applicazioni nel commercio elettronico, la cooperazione, ecc.

Per poter affermare che due identificatori diversi fanno in realtà riferimento alla stessa risorsa si utilizzano le ***ontologie.***

2.6 Ontologie

Le *ontologie* rappresentano l'elemento alla base di tutto il Web Semantico.
In ambito filosofico una Ontologia indica lo studio dell'*essere* in quanto tale e delle sue categorie fondamentali.
Spostandoci al contesto informatico possiamo notare come la sostanza della definizione resti intatta, seppur con alcune differenze: una ontologia è la descrizione di concetti, delle relazioni tra di essi e delle loro proprietà. In termini semplici, si potrebbe dire che una ontologia permette di rappresentare le entità tramite la descrizione delle loro caratteristiche e le relazioni che intercorrono tra di loro o tra domini della conoscenza; si può quindi definire come un sistema di metadati che fanno riferimento a determinati vocabolari (schemi). Ad esempio, lo Schema RDF definito al § 2.4 può essere considerato con buona approssimazione un'ontologia definita tramite l'RDFS stesso.
Tramite le ontologie diventa possibile fare inferenze, tracciare relazioni e deduzioni sui dati, che vengono così ad assumere valore semantico. Un dataset, ovvero una collezione di risorse relative ad un qualche ambito (vedi Figura n. 2), è quindi anch'esso definito e descritto da una ontologia.

[11] da http://www.websemantico.org/articoli/approcciwebsemantico.php#quattro

Una delle definizioni più rigorose è quella di Gruber secondo cui:
"*una ontologia è una specificazione esplicita di una concettualizzazione*"[12]

Una *concettualizzazione* è una visione astratta di un particolare ambito: volendo descrivere o gestire un ambito della conoscenza si dovrà pertanto fare ricorso ad una descrizione astratta del particolare dominio di interesse, ovvero bisognerà darne una definizione dei concetti e delle relazioni tra di essi.
Un buon esempio di ontologie può essere quello delle categorie utilizzate nella catalogazioni dei libri in biblioteca.
Un'ontologia è composta da un elenco di nomi insieme alle relazioni di tipo tassonomico (tipo-sottotipo) che intercorrono tra di loro; consente pertanto di strutturare le informazioni sul Web.
I vantaggi delle ontologie sono dunque diversi ed applicabili in vari campi. Nel caso di informazione proveniente da più sorgenti e in diversi formati, le ontologie forniscono una descrizione formale dei dati in modo da semplificare il lavoro di ricerca degli stessi. Ciò risolve il problema della mancanza di struttura, propria del Web attuale, e fornisce degli strumenti per disambiguare il contesto di utilizzo di un particolare termine, evitando possibili confusioni.
Il livello Ontologico del Web Semantico è oggetto di studio e di sviluppi continui in quanto centrale nell'ambito del "*Knowledgeable Web*", in cui la gestione e il trattamento dell'informazione siano interamente di competenza delle macchine.
A tal proposito nel 2001 Tim Berners Lee affermava in un articolo[13]:

"*Per permettere il funzionamento del Web Semantico, i computer devono avere accesso a collezioni strutturate di informazioni e a set di regole di inferenza che possano usare per condurre un ragionamento automatizzato. I ricercatori dell'intelligenza artificiale hanno studiato tali sistemi a partire da molto tempo prima che il Web fosse sviluppato. La rappresentazione della conoscenza […] attualmente è in uno stato paragonabile a quello dell'ipertesto prima dell'avvento del Web: è chiaramente una buona idea, ed esistono alcune dimostrazioni molto*

[12] T.Gruber, (1993), *A translation approach to portable ontology specification*, www.tomgruber.org

[13] Berners-Lee T., Hendler J., Lassila O., *Op. cit.*

valide, ma non ha ancora cambiato il mondo. Contiene i semi di applicazioni importanti, ma per realizzare il suo pieno potenziale deve essere collegata all'interno di un singolo sistema globale."

Lo sviluppo delle ontologie renderebbe dunque le macchine capaci di ragionare per inferenza.

Una prospettiva in questo senso è delineata dallo stesso Berners Lee nell'articolo del 2001, più volte citato.
Viene fatto l'esempio di due fratelli: Lucy e Pete che si trovano a dover fissare un appuntamento di fisioterapia per la madre. L'agente Web Semantico di Lucy, incluso nel suo browser, fa una ricerca di tutti gli specialisti che effettuino quel tipo di prestazione medica e sceglie in modo completamente autonomo quelli che si trovano in un raggio di 40 chilometri da casa e che godono di buona fama. Inoltre fa un controllo incrociato dei posti disponibili per la visita e degli impegni di Lucy e Pete. In pochi minuti l'agente Web Semantico fornisce, in modo del tutto automatico, i risultati. I due fratelli possono accettare o rifiutare inserendo ulteriori vincoli di orario o di distanza.

2.6.1 I linguaggi delle Ontologie (OIL, DAML, OWL)

Lo sviluppo del livello Ontologico dipende in larga parte dai linguaggi che definiscono le Ontologie.
Nel Web semantico abbiamo il livello dei dati, di cui si occupano XML e RDF, e l'RDF Schema, utile per la definizione dei vocabolari. Questi strumenti non sono però sufficienti a catalogare le informazioni dal punto di vista semantico.
Ecco che a questo punto entrano in gioco i *linguaggi per la creazione delle ontologie* che costituiscono il livello superiore nel modello piramidale del Web Semantico (vedi Figura n.1).
Attualmente possiamo citare ***OIL (Ontology Inference Layer), DAML (DARPA Agent Markup Language+OIL) e OWL (Web Ontology Language)***
Essi sono linguaggi di marcatura per esplicitare il significato dei termini e le loro relazioni; sono basati su RDF Schema di cui costituiscono un'estensione.
I sistemi di rappresentazione della conoscenza tradizionali permettono di accedere ai

dati con query e domande "fisse" senza possibilità di variazione dalla forma standard con la quale il contenuto è memorizzato, pena l'impossibilità di rispondere da parte del sistema.
Il Web Semantico dovrebbe sopperire proprio a questo limite della rappresentazione della conoscenza tradizionale. Ciò si può raggiungere tramite le ontologie e il livello logico (ancora in fase di sviluppo) che permetterà di fare inferenze sui dati.
Proprio a questo scopo, tramite il linguaggio OWL, è possibile associare una proprietà semantica a determinati concetti ed essere in grado di fare assunzioni e deduzioni su di essi evitando omonimie e ambiguità.

Vediamo un esempio di Ontologia descritta in OWL.

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="food;PotableLiquid"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#madeFromGrape"/>
      <owl:minCardinality
rdf:datatype="nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

da http://www.w3.org/TR/2004/REC-owl-guide-20040210/#owl_equivalentClass

Possiamo notare il fatto che ci siano molte analogie con lo Schema RDF presentato al § 2.4, ma come in questo caso OWL espanda classi e proprietà.
Nell'ontologia viene specificato che la *classe* "Wine" è una *sottoclasse* della classe "food;PotableLiquid" e di un'altra generica classe denominata "Restriction" che rappresenta entità con *almeno una* ("minCardinality") proprietà "madeFromGrape".
Dal momento che "Wine" è dichiarato come sottoclasse della restrizione, ne deriva che tutte le entità Wine hanno a loro volta le *proprietà* di "Restriction".
Con questo semplice esempio si è esplicitata una descrizione comprensibile anche alle macchine del significato di una determinata risorsa (*classe*). Essa può essere adesso utilizzata per ricavare nuova informazione, o per integrare quella già esistente (rimanendo all'esempio visto, la sottoclasse "Wine" può essere combinata con altre entità della classe "food").

3. Dai bollettini di guerra ai linked data

3.1 Uno sguardo d'insieme

Il progetto è frutto di una collaborazione congiunta tra l'Istituto di Linguistica Computazionale "Antonio Zampolli" del CNR di Pisa e del CoLing Lab, Dipartimento di Filologia, Letteratura e Linguistica dell'Università di Pisa.
L'obiettivo è l'elaborazione computazionale di un corpus storico di dispacci prodotti nei due conflitti mondiali, visto che ancora non esiste un progetto in questo senso e considerato che nel 2014 è caduto il centenario dell'inizio della Prima Guerra Mondiale.
Si può infatti affermare che la Prima e la Seconda Guerra Mondiale rappresentino tappe fondamentali nel XX secolo e nella storia dell'Occidente. Lo scopo era dunque quello di rendere questi eventi accessibili ad una vasta fetta di pubblico, sradicando un cliché che vede la storia come appannaggio di studenti ed esperti.
Per donare freschezza ed una maggiore fruibilità ad eventi tanto importanti e, se ben trattati, appassionanti, è stata presa la decisione di servirsi delle moderne tecnologie informatiche. Sono stati quindi presi in esame i documenti che forse meglio di altri "fotografano" la situazione bellica giorno per giorno e che ci restituiscono una visione di primo piano su eventi e protagonisti: stiamo parlando dei ***bollettini di guerra***.
Essi vennero pubblicati giornalmente sui quotidiani dell'epoca, trasmessi alla radio (durante la sola IIGM) e documentano gli eventi che vanno **dal 24 maggio 1915 all'11 novembre 1918** per la IGM (1361 bollettini) e **dal 10 giugno 1940 al 8 settembre 1943** (data della firma dell'armistizio) per la IIGM (1201 bollettini).
Essendo redatti dal "Comando supremo"[14], risentono dell'enfasi della propaganda, tacendo o sottostimando alcuni eventi "scomodi" e rimarcando gli aspetti positivi. In tal modo è stato possibile studiare questi preziosi documenti non solo dal punto di

[14] Organo militare del Governo italiano

vista della storia militare, ma anche per quel che riguarda il trattamento delle notizie da parte di un organo ufficiale.
I bollettini della IGM sono stati pubblicati per la prima volta nel 1923, mentre quelli della IIGM sono disponibili dal 1970 e sono stati recentemente digitalizzati e resi disponibili in formato html[15].
I due corpora di bollettini (I e II GM) sono stati analizzati insieme: oggigiorno infatti gli storici concordano nel definire le due Guerre mondiali come parti di uno stesso capitolo della storia europea della durata di un trentennio. Inoltre la loro analisi comparativa mette in luce le differenze sul piano militare e politico nei due conflitti, differenze che hanno ripercussioni sul piano linguistico e propagandistico.
Il progetto seguito dal CoLing Lab e dal CNR ha avuto come step principali:

1. **Digitalizzazione dei bollettini della IGM (per la II GM si è lavorato su un corpus già digitalizzato).**
2. **Annotazione mediante *strumenti per il trattamento automatico della lingua* (TAL)**, con i quali il corpus è stato sottoposto a **POS-tagging**[16]**.**
3. **Analisi statistica ed estrazione delle informazioni,** con la quale i testi sono stati indicizzati con informazioni linguistiche e semantiche.
   - Ad ogni bollettino sono stati associati alcuni dati statistici come il numero di token, la type token ratio ecc. che ci permettono di ottenere interessanti informazioni anche sulla lunghezza dei bollettini stessi, derivata dal resoconto di eventi differenti.
   - Dal corpus sono state estratte anche alcune parole significative che poi andranno a costituire utili parole chiave per le ricerche.
   - Le entità nominate (named entities) sono state recuperate e divise in alcune categorie: LOC per i luoghi, PER per i nomi propri di persona, SHP per i nomi di nave, PLN per i nomi d'aereo e infine MIL per i termini propri militari.
   - Infine verranno **annotati gli eventi bellici** che possiamo ricavare dalla lettura dei bollettini (bombardamenti, affondamenti, perdite ecc.), insieme ai protagonisti, al tempo e al luogo dell'avvenimento. Così facendo verrà realizzata una vera e propria *timeline* degli eventi bellici.

[15] http://www.alieuomini.it/pagine/dettaglio/bollettini_di_guerra
[16] Annotazione delle parti del discorso

4. **I dati** così estratti sono stati **collegati a risorse esterne**; i nomi di luoghi sono stati ***georeferenziati*** in modo da sopperire ad alcune ambiguità determinate dal diverso nome che uno stesso sito assume in lingue diverse, o causate da località che dopo gli eventi bellici sono entrate a far parte di Stati confinanti (es. Italia e Slovenia). **Le entità nominate sono state inoltre collegate ad altre risorse esterne, come ad esempio Wikipedia.**
5. Come ultimo passo il lavoro è stato reso disponibile per mezzo di un'**interfaccia**[17] che è in grado di offrire un supporto anche ai ricercatori più esperti.

   Essa dovrebbe essere in grado di permettere ricerche per singola parola, parole composte, classi semantiche, eventi, luoghi, persone ecc. È inoltre prevista una tag cloud con le parole chiave, una timeline con gli eventi principali, ed una mappa in cui siano questi siano evidenziati.

Tutti i bollettini finora acquisiti sono stati digitalizzati tramite OCR (Optical Character Recognition) utilizzando il software open source *Tesseract*.
Parallelamente alla digitalizzazione dei bollettini della I GM, sono stati condotti esperimenti sull'annotazione linguistica automatica dei bollettini della Seconda Guerra Mondiale, tramite strumenti per il TAL[18]. Il corpus di bollettini della II GM è stato scaricato e ripulito dei tag HTML, annotato automaticamente con il POS tagging e infine sono state annotate le dipendenze sintattiche.
I bollettini delle due Guerre Mondiali presentano caratteristiche linguistiche proprie dell'italiano della prima metà del Novecento e in molti casi un lessico di tipo tecnico. Le frasi sono più corte che sui quotidiani ed inoltre non mancano ellissi ed omissioni dovute alla brevità richiesta dalla trasmissione via telegrafo. A ciò si aggiunge il fatto che l'annotazione linguistica può risentire di espressioni linguistiche desuete: l'italiano della prima metà del XX secolo era infatti piuttosto differente da quello attuale, principalmente perché era ancora in via di sviluppo, dato che l'Italia era stata unificata appena mezzo secolo prima.
Pertanto, nonostante gli strumenti per il TAL siano particolarmente affidabili e precisi, sono stati riscontrati diversi errori in fase di POS tagging.

[17] http://www.memoriediguerra.it/

[18] http://www.ilc.cnr.it/?q=it/content/trattamento-automatico-del-linguaggio-naturale-ed-estrazione-di-conoscenza

Poiché i bollettini riportano eventi bellici, essi sono **ricchi di nomi propri** di luoghi, persone, enti ecc., che rivestono un ruolo molto importante nell'opera di analisi dei bollettini. Per questo ne viene tenuta traccia mediante il campo **NER** (*Named Entity Recognition*) che permette un miglior accesso *semantico* al corpus.
Le entità nominate sono identificate tramite un classificatore denominato *CoLing Lab NER* adattato ai corpora dei bollettini. Si tratta di un NER per la lingua italiana, sviluppato con lo Stanford CoreNLP NER[19] e addestrato su I-CAB (Italian Content Annotation Treebank), un corpus di notizie provenienti dal quotidiano "L'Adige" e annotato con informazioni semantiche. Nel corpus I-CAB sono state identificate 4 tipologie di NE: LOC (Locations), GPE (Geo-Political Entities), ORG (Organizations), PER (Persons).
Come corpus di prova per l'addestramento del NER (Named Entity Recognizer) ai bollettini di guerra è stato scelto quello relativo ai bollettini della IGM[20]. Per avere una versione *gold standard* in breve tempo, il corpus è stato digitalizzato, annotato in modo semiautomatico utilizzando il NER esistente, e successivamente controllato manualmente da un utente umano. In questo caso la tagset del NER consiste di 5 classi: LOC (Locations), PER (Persons), MIL (Military Organizations), SHP (Ships), PLN (Plane).
L'adattamento del NER al corpus di bollettini della IGM è avvenuto mediante 3 metodi differenti che sono stati poi confrontati:

1. **I bollettini sono stati annotati utilizzando il NER del CoLing Lab addestrato su una versione modificata di I-CAB.**
   Le NE di tipo LOC sono state combinate con quelle di tipo GPE; inoltre le ORG di I-CAB sono state mappate sulle NE di tipo MIL.
2. **Il corpus I-CAB è stato combinato con WB2** (corpus annotato, creato a partire dalla versione digitalizzata dei bollettini della IIGM[21]).
   È stato creato un corpus di addestramento partendo dalla versione HTML dei bollettini della II GM che ha il grosso pregio di avere un indice analitico dei nomi[22] (NE) presenti nel testo. Quindi il corpus è stato ripulito dei tag HTML, annotato con le stesse 5 categorie di NE descritte in precedenza e,

[19] http://nlp.stanford.edu/software/CRF-NER.shtml
[20] *I Bollettini della Guerra 1915-1918*, (1923), prefazione di Benito Mussolini, Milano, Alpes
[21] http://www.alieuomini.it/pagine/dettaglio/bollettini_di_guerra
[22] http://www.alieuomini.it/pagine/dettaglio/bollettini_di_guerra,9/indici,92.html

visti i buoni risultati, è stato mappato sulle classi di I-CAB ed utilizzato come gold standard per annotare il corpus della IGM

3. **Il corpus della IGM è stato annotato con il NER addestrato su WB2.** In quest'ultimo caso i testi confrontati sono vicini sia a livello cronologico che per quanto concerne i tipi di NE (soprattutto PER e MIL); è quindi il metodo che ha dato i risultati migliori.

Per l'individuazione delle entità nominate (o *named entities*, NE), i criteri con cui il NER è stato addestrato sul corpus WB1 sono gli stessi per tutti i tre metodi elencati: **caratteristiche morfologiche e ortografiche, informazioni sulla forma dei termini**, **POS-tag**, **tag delle entità nominate** e **caratteristiche di contesto**; in particolare:

- La **caratteristica delle parole**: rispetto ad un termine di riferimento, sono stati considerati il precedente e il seguente (ad esempio in "Colonnello Piccio", la prima parola, "Colonnello", aiuta ad identificare la seconda come entità di tipo PER). Un altro metodo consiste invece nell'individuare ed analizzare le 3 parole che precedono e seguono un determinato termine di riferimento.
- **Caratteristiche ortografiche**: scansione delle parole, lettere maiuscole, presenza di caratteri non alfabetici, ecc.
- **Caratteristiche linguistiche**: posizione delle parole nella frase (determinata da un attributo numerico), il lemma e il POS-tag (determinati da attributi nominali).
- I **Termini complessi** estratti con l'algoritmo EXTra (es. "capitano di corvetta") sono stati identificati come un unico termine ed hanno concorso anch'essi all'addestramento del modello.

Il progetto prevede la completa digitalizzazione dei bollettini della I GM, insieme alla correzione manuale dei corpora delle due Guerre. Le entità nominate di tipo geografico dei due corpus sono state *georeferenziate* ed è in corso lo sviluppo di un insieme di metodologie per *l'estrazione degli eventi* dai corpora di riferimento.
In sostanza, come vedremo in seguito, alla fine del processo di analisi si sono

ottenuti due corpora di bollettini rilasciati come **triple RDF** con varie informazioni linguistiche e semantiche.

3.2 Linking dei bollettini a Wikipedia

Partendo dai bollettini delle due Guerre Mondiali digitalizzati e arricchiti con tutte le informazioni e i tag visti in precedenza, si è giunti ad avere una versione tokenizzata e POS-taggata[23] dell'intero corpus dei bollettini della I GM insieme a quelli della II GM.
Il processo di lemmatizzazione e annotazione delle parti del discorso e delle dipendenze sintattiche è stato raccolto in un file ***CONLL***[24] in cui il formato di ogni riga è il seguente:

Token ID	Forma	Lemma	POS-Tags	NER	Disamb.	URL WP

Per esempio:

9	Sicilia	Sicilia	S	SP	_	8	prep	B-LOC	O	O

In sequenza abbiamo *token id*, *forma*, *lemma* (ovvero la forma normalizzata), i tag che annotano *le parti del discorso* e le *dipendenze sintattiche* . In terzultima posizione il tag *NER* (Named Entity Recognition) che appunto tiene traccia delle *entità nominate*, ovvero i nomi propri. Troviamo poi il campo utilizzato per la disambiguazione ed infine quello in cui verrà stampata l'*URL Wikipedia* (se trovata).
Il primo punto che abbiamo affrontato nello svolgimento del progetto, riguarda lo studio di metodologie per l'estrazione di URL Wikipedia associate alle entità nominate contrassegnate dal tag NER.
Questo campo, può assumere fondamentalmente due possibili valori:
- Il **valore "O"** (*Outside*) quando non siamo in presenza di una NE, ovvero nel caso di nomi comuni, aggettivi, verbi, articoli ecc.

[23] http://medialab.di.unipi.it/wiki/Tanl_POS_Tagset
[24] Conference on Natural Language Learning

- Il **valore ≠ "O"**, se siamo in presenza di una NE

In caso di valore ≠ "O", il campo NER può assumere i seguenti valori:

- LOC=Location, Luogo
- PER=Person, persona
- SHP=Ship, nave
- PLN=Plane, aereo
- MIL=Military, ambito militare

dove l'entità nominata può essere il nome proprio di un luogo, di una persona, di una nave, di un aereo o un termine di ambito militare.
La prima parte dell'entità nominata (eventualmente l'unica dell'entità, es. "Africa" o "Valle Lagarina") avrà il "prefisso" B- (*Beginning*) nel campo NER. Se l'entità è composta da più parole adiacenti (es.Valle Lagarina, Marsa Matruh ecc.) dal secondo termine in poi il NER avrà "prefisso" I- (*Inside*).

Facciamo un esempio per comprendere meglio:

33	Africa	Africa	S	SP	_	31	mod	B-LOC	O	O
2	valle	valle	S	S	num=s|gen=f	1	prep	B-LOC	O	O
3	Lagarina	Lagarina	S	SP	_	5	subj	I-LOC	O	O

Nell'ambito del Web semantico gli URI giocano un ruolo fondamentale in quanto ogni risorsa è individuata univocamente per mezzo di essi. Pertanto anche il nostro corpus di bollettini delle Guerre Mondiali necessitava di identificatori che riuscissero ad assegnare un significato univoco alle entità nominate e con la minor ambiguità possibile. Per questo motivo alle entità nominate è stato assegnato un contenuto semantico associando ad esse, direttamente nel file CONLL, l'**URL della pagina Wikipedia corrispondente**.
Conferendo un potere semantico alle parole chiave, tramite gli URL dell'enciclopedia on-line per eccellenza, si riesce a realizzare lo scopo, non secondario, di far divenire il corpus un *ipertesto* fruibile da studiosi ed appassionati

dell'argomento e a conferirgli quindi *proprietà semantiche*.
Si potrebbe obiettare che Wikipedia non abbia un alto grado di affidabilità; tuttavia essa presenta numerosi vantaggi, primo fra tutti la forma particolarmente semplice e standardizzata degli URL che identificano le sue voci. Oltre a ciò, Wikipedia ha una diffusione che la rende capace di coprire una vasta gamma di ambiti e di tematiche, rendendola particolarmente adatta ai nostri scopi.
Va inoltre aggiunto che nel nostro corpus di riferimento le parole, oggetto di ricerca nell'enciclopedia on-line, appartengono ad ambiti che prestano il fianco in misura molto ridotta ad interpretazione soggettive (si pensi per esempio ai nomi di luogo).
Infine è bene sottolineare che il lavoro in oggetto vuole solo dare un saggio delle grandi potenzialità insite nel Web Semantico e di ciò che diventa possibile ottenere associando ad un corpus proprietà semantiche; sono quindi possibili, ed in qualche modo auspicabili, modifiche e sviluppi del modello proposto.
Il formato CONLL permette di avere tutte le informazioni utili per le ricerche linguistiche ma si presta molto bene ad essere trattato da un linguaggio di programmazione. Nel nostro caso si è scelto di utilizzare *Python*[25], un linguaggio di programmazione *orientato agli oggetti* dalla sintassi snella, ma dalle grandi potenzialità.
Ogni riga del file CONLL è vista da Python come una *lista*[26], rendendone in tal modo semplice ed immediata l'elaborazione.
Una volta specificato un file in input, con Python è possibile scansionarlo riga per riga tramite un banale *ciclo for*, applicando a ciascuna il metodo `split(<sep> )` che restituisce una lista delle parole contenute nella stringa, usando `<sep>` come delimitatore.
Nel nostro caso il delimitatore `<sep>` è un carattere di tabulazione "**`\t`**" che ci permette di tenere ben separati i vari tag del file CONLL e allo stesso tempo di fare riferimento a loro mediante la sintassi per accedere agli elementi di una lista:

```
Lista[i]
```

[25] http://www.python.it

[26] http://docs.python.it/html/tut/node5.html#SECTION005140000000000000000

Per ogni singola riga del file CONLL (ad eccezione di quelle vuote) è stata scelta la codifica UTF-8 (Unicode Transformation Format, 8 bit)[27] di UNICODE[28], in modo da non avere problemi di sorta con i caratteri speciali (soprattutto accenti e dieresi), visto che la codifica predefinita di Python è quella ASCII.
Python mette inoltre a disposizione la funzione built-in `unicode( )`; essa permette l'accesso a tutti i *codec* in grado di convertire le stringhe nelle codifiche utilizzate più spesso (Latin-1, ASCII, UTF-8 e UTF-16).
In sostanza ogni riga è stata codificata e convertita utilizzando la sintassi:

```
uLine = unicode(line.strip(),"utf8")
```

Inoltre tutti gli output sono stati codificati in UTF-8 tramite il metodo `encode( )`:

```
outfile.write(outstring.encode("utf8"))
```

Vediamo i vari passi che ci hanno permesso di strutturare il corpus di bollettini con tag semantici:

1. **Il primo passo ha previsto l'estrazione di tutte righe del file CONLL in cui erano presenti le entità nominate (NE) mediante lo script open_file_Conll.py** (vedi appendice codici n. 1).

Lo script prende in input il file CONLL originario:

```
input_file = bollettiniGG.ner.txt
```

per la IGM

oppure

```
input_file = bollettini_IIGM.ner.txt
```

per la IIGM

e restituisce in output le righe in cui siano presenti le entità nominate, ovvero quelle per cui valga la condizione:

```
 if ner != "O":
```

[27] http://it.wikipedia.org/wiki/UTF-8
[28] http://www.unicode.org/

L'output è risultato della forma:

206 2 Mediterraneo Mediterraneo S SP _ 1 prep B-LOC

2. **Il passo successivo è consistito nell'estrazione delle entità nominate** avendo cura di unire quelle composte da più sottoparti, ma costituenti un unico token (es. Valle Lagarina), in cui il NER con prefisso B- è stato concatenato con una o più sottoparti con prefisso I-, ognuna separata dal carattere "_".

Ciò è stato fatto mediante lo script extract_entities.py (vedi appendice n.2) di cui si riporta un estratto:

```
if ner.startswith("B-"):
  ner_lemma_s = spl_line[1].lstrip("-").capitalize()
elif ner.startswith("I-"):
  ner_lemma_s += "_" + spl_line[1].lstrip("-")
```

Per esempio: Valle →B-LOC; Lagarina →I-LOC

è diventato:

Valle_Lagarina

Come output si è ottenuto un file .indexes di *mapping* composto da tante righe quante sono le entità nominate di quel determinato corpus di riferimento. Ogni riga del file contiene tre campi estratti direttamente dal CONLL di cui costituisce una proiezione:

Token id, Tipo di NER, Forma (la NE)

Ad esempio:

578 LOC Italia

3. A questo punto del lavoro è stata valutata la possibilità di scaricare il dump rilasciato da Wikipedia e di interrogarlo tramite Python con SQL. Tuttavia si è preferito agire in maniera più semplice, ma sostanzialmente più efficace**, concatenando alla forma base dell'URL Wikipedia http://it.wikipedia.org/wiki/** il terzo l'elemento del file CONLL, ovvero **il lemma**, ottenendo una URL del tipo:

http://it.wikipedia.org/wiki/NamedEntity

vediamo un estratto del codice per capire meglio.

La forma base dell'URL Wikipedia è stata assegnata alla variabile `wiki_cost`

```
wiki_cost = "http://it.wikipedia.org/wiki/"
```

concatenando ad essa la NE estratta direttamente dal file .indexes ed ottenendo dunque una URL Wikipedia.
Ad esempio:

206 2 Mediterraneo Mediterraneo S SP _ 1 prep B-LOC

Ed il codice corrispondente (tratto da stampaurl.py, vedi appendice n.3).

```
lemma=spl_line[2]
search_str = "_".join(spl_line[2:])
#con spl_line[2:] nel file CONLL prendo la forma
```

Che è stato poi aggiunto in coda all'URL base di Wikipedia:

```
test_url = wiki_cost + search_str
```

A questo punto si è proceduto al controllo dell'esistenza delle URL di cui sopra mediante il costrutto:

```
try
except
```

di Python, invocando il modulo:

```
urllib2.urlopen(test_url)
```

per verificare se la pagina esistesse, o meno:

```
except urllib2.HTTPError, err:
if err.code == 404:
```

Lo script stampaurl.py ha ricevuto in input il file .indexes di cui si è parlato al punto 2, ricavando in output un file contenente:

Token id, Tipo di **NER,** e **URL Wikipedia**
nel caso di pagina WP trovata.

Oppure:
Token id, Tipo di **NER**, e **carattere 'O'**
nel caso di pagina WP non trovata.

Si è osservato che nei vari corpus il rapporto tra pagine WP trovate e non trovate è il seguente[29]:

I GM → 7441 FOUND / 5867 NOT FOUND

II GM → 11266 FOUND / 1785 NOT FOUND

[29] Analisi eseguite con gli script contaEntità.py, contaTokens.py, contaUrl.py, (vedi Appendice codici 4a, 4b, 4c)

Vediamo nel dettaglio:

		1915	1916	II GM
Numero di tokens del corpus		42959	54688	211851
Numero di entità (differenziato per tipo)	LOC	2394	4303	8504
	MIL	9	59	132
	PER	4	0	114
	SHP	33	1	140
	PLN	5	32	372
Numero di url trovati (differenziati per tipo)	LOC	1530	2314	7609
	MIL	4	21	24
	PER	2	0	46
	SHP	26	0	90
	PLN	3	29	344
Numero totale di entità		2445	4395	9262
Numero totale di url trovati		1565	2364	8113

Figura n. 6

4. Infine **tramite lo script conll_urls.py** (vedi appendice codici n.5) **che accetta in input il file di mapping con Token ID, NER e URL di cui sopra, insieme al file CONLL originario, si è generato in output un file CONLL in cui accanto alle Named Entities si hanno gli URL Wikipedia relativi.**

Vediamo in dettaglio lo script conll_urls.py che è composto da due funzioni principali:

- `def urls2conll(ent_ind_file, conll_file):`
  che estende il CONLL con le URL di Wikipedia.
- `def coordinate (coord_file, extended_conll):`
  per stampare le coordinate geografiche nel file CONLL

La funzione `def urls2conll(ent_ind_file, conll_file):`
accetta in input il file di mapping con token id, NE ed URLWP e il file CONLL originario.
Si parte inserendo in un *dizionario* Python la token id come chiave e la URLWP

corrispondente come valore; il numero di entrate {token id: URLWP} è determinato dal numero di parti della NE.
Ad esempio "San_Giusto" è una NE composta da 2 parti e dunque nel dizionario si avrà:

{105:http://it.wikipedia.org/wiki/San_Giusto,
106:http://it.wikipedia.org/wiki/San_Giusto}

Successivamente è sufficiente confrontare la chiave (token id) con quella del file CONLL originario e, nel caso siano uguali, stampare l'URL Wikipedia in ultima posizione nel file CONLL stesso:

```
if idx_dict.has_key(idx):


substring = "\t".join(idx_dict[idx])
outstring = line.strip() +  "\t" + substring + "\n"
```

quando l'URLWP esiste; si stampa invece 'O' quando non è stata trovata.

L'output di questa prima parte dello script pertanto è un file CONLL della forma:

30 San San S SP _ 31 mod B-LOC O
http://it.wikipedia.org/wiki/San_Giusto

31 Giusto Giusto S SP _ 29 prep I-LOC O
http://it.wikipedia.org/wiki/San_Giusto

3.2.1 Casi in cui l'URL di Wikipedia non è stato trovato e possibili soluzioni

Analizzando i corpus si è visto che i casi di NOTFOUND dipendevano principalmente da due motivi:

- **Pagina Wikipedia** sull'argomento semplicemente **non esistente**

- **Problemi di ambiguità** che si sono presentati per i motivi più disparati, spesso anche in ambiti apparentemente "poco probabili".

Ecco alcuni casi di ambiguità e NOT FOUND con possibili metodologie per poterli risolvere:

a. Spesso **uno stesso toponimo designa località diverse** sparse nella Penisola, in Europa o nel mondo. Questo tipo di ambiguità è il più difficile da risolvere, perché viene catalogato dallo script come "FOUND", mentre in realtà il link di Wikipedia punta ad una pagina di disambiguazione.
b. **Alcune entità**, sempre di tipo LOC, **sono presenti nei corpora con il nome in lingua tedesca o slovena, mentre su Wikipedia compaiono con il corrispondente italiano**, o con il nome assunto dopo i cambiamenti successivi agli avvenimenti bellici. Si viene così a generare il NOT FOUND di una voce che in realtà esiste. Es. Val_Fischlein → *Val_Fiscalina*
c. Si sono poi presentati casi di **ambiguità relativamente ad entità di tipo LOC che designano anche nomi di cose**. Per es. Volàno, Tùrbine ecc.
   Per questo genere di problemi è stata decisiva la scelta di mappare nell'output finale (rilascio del corpus come Linked Data, vedi Cap. 3) ogni NE con il suo NER ed un numero progressivo che la identificasse in modo univoco.
d. Esistono poi **NOT FOUND "nascosti"** nel senso che l'URL risulta trovato ma punta ad una **pagina di disambiguazione** in cui può essere o meno presente l'ambito che ci interessa. (es. San_Martino, che si può riferire a diverse località, oltre che al santo)
e. **Alcune entità sono risultate NOT FOUND poiché presenti su WP con articoli o preposizioni diverse**. Ciò è dovuto al fatto che, andando ad operare sugli URL di Wikipedia, ogni piccolo scostamento dalla forma può avere grande rilevanza.
   (es. S._Martino_del_Carso risulta non trovato in quanto l'URL di Wikipedia prevede San _Martino_del_Carso).
f. In taluni (pochi) casi anche se l'entità non fa esattamente match con la voce presente nell'url wikipedia, ci viene in aiuto la "nota di reindirizzamento". (es. San_Donà reindirizza automaticamente a San_Donà_di_Piave)
g. Ci sono poi **toponimi** che nel corpus sono abbreviati o sono **presenti con aggettivi che impediscono il recupero su WP**: es. "Gradisca", che su WP si trova come *Gradisca_d'Isonzo*, oppure "Medio Isonzo" che ovviamente su WP è presente solo come *Isonzo*.

h. Alcuni **NOT FOUND** sono stati **determinati da tokenizzazioni errate,** del tipo "Settore_di_Tolmino" anziché annotare come entità esclusivamente "Tolmino" e quindi incollare quello nell'URL di WP.

Per risolvere alcuni dei problemi di ambiguità visti sopra, lo script è stato reso in grado di estrarre le coordinate geografiche (latitudine e longitudine) direttamente dal file "luoghi.csv", fornito dal Laboratorio di Linguistica computazionale.
In esso sono presenti 8 campi:

Contatore progressivo delle NE di tipo LOC
Guerra: I o II GM
Luogo 1: NE di tipo LOC
Luogo 2: Informazioni utili per la disambiguazione: varianti toponomastiche del Luogo 1, eventualmente insieme al nome dello Stato di cui fa parte ed al suo CAP
Tipologia di luogo: city, town, establishment, ecc.
Latitudine, Longitudine: le coordinate geografiche
Luogo 3: Ulteriori varianti toponomastiche (ove necessario)

Le Coordinate Geografiche (Latitudine e Longitudine) di cui si parlerà ulteriormente nel cap. 3, e le varianti toponomastiche, hanno contribuito in modo decisivo a risolvere i problemi descritti, almeno per i punti a, b, c, d, g visti poc'anzi.
In questo modo è stato stampato un unico file CONLL "esteso" in cui oltre ai campi del CONLL originale (token id, forma, lemma, dipendenze, NER) si avessero anche URL Wikipedia e Coordinate Geografiche.

Vediamo la descrizione di questa seconda parte dello script conll_urls.py

La seconda delle due funzioni elencate:
`def coordinate (coord_file, extended_conll):`
prende in input il file delle coordinate (luoghi.csv) e restituisce in output il file CONLL esteso.
Si è dapprima creata una *tupla* con tutti i campi del file csv visti:

```
Num_progr,guerra,LOC_Name1,LOC_Name2,LOC_Type, LAT, LONG,_ =
uLine
```

I Luoghi 1 e 2 sono stati poi inseriti in una *lista*, inserendo come separatore il carattere `'_'` per permettere il confronto con le NE presenti nel file di mapping (che contiene token id, NE ed URLWP) , visto che nel file CSV le NE erano prive di tale separatore.

```
luoghi=[LOC_Name1.replace(' ', '_'), LOC_Name2.replace(' ',
'_')]
```

Infine i dati presenti nel file delle coordinate vengono memorizzati in un *dizionario* Python, in cui le NE sono le *chiavi* e le coordinate i *valori*.

```
for luogo in luoghi:

            dict_coord[luogo] = [LAT,LONG]
```

Dopo di che vengono confrontate le varie entità presenti nel dizionario con quelle presenti nel file CONLL; se c'è corrispondenza, nelle ultime due posizioni del file CONLL esteso vengono stampate le coordinate geografiche (latitudine e longitudine), altrimenti vengono stampati nelle stesse posizioni due stringhe "Null".

```
if dict_coord.has_key(NER_forma_s):

                        outstring =  line.strip()+"\t"+
"\t".join([a for a in dict_coord[NER_forma_s]])

                    else:

                        outstring =  line.strip()+"\t"+
"Null\tNull"
```

Riprendendo l'esempio fatto all'inizio e tenendo conto di tutto il codice visto, l'output finale sarà:

9 Sicilia Sicilia S SP _ 8 prep B-LOC O http://it.wikipedia.org/wiki/Sicilia Null Null

nel caso in cui nel file delle coordinate non sia presente l'entità.

Oppure:

12	Napoli Napoli S	SP	_	11	prep	B-LOC	O
	http://it.wikipedia.org/wiki/Napoli			40.85177612	14.26812363		

in caso contrario.

3.3 Generazione dei Linked Data

Lo sviluppo del nostro progetto ha quindi previsto il rilascio dei bollettini come triple RDF, permettendo in tal modo di conferire alle risorse proprietà semantiche tali da permetterne l'interrogazione sotto vari punti di vista.
Il primo passo ha previsto la definizione di uno *schema RDF*[30] per definire classi di risorse e proprietà che le definiscano (vedi figura n.3, §2.3):

```
<?xml version="1.0"?>
<rdf:schema xmlns:rdf="http://www.w3.org/2001/XMLSchema"
            xmlns:dc="http://purl.org/dc/elements/1.1/">
    <rdf:Description rdf:about="URI bollettino=numero(index)_data">
        <dc:source>URL bollettino = (es. http://www.memoriediguerra.it/bollettini/37749)
            <s:index>index bollettino
            <dc:creator>firma bollettino </dc:creator>
               <rdf:Description rdf:about="NERType_NumNE">
                <s:PER>Persone
                  <s:URLWP>URLWP</s:URLWP>
                </s:PER>
                <s:LOC>Luoghi
                  <s:Geo>coordGeo</s:Geo>
                  <s:URLWP>URLWP</s:URLWP>
                </s:LOC>
                <s:MIL>Entità Militari
                  <s:URLWP>URLWP</s:URLWP>
                </s:MIL>
                <s:PLN>Aerei
                  <s:URLWP>URLWP</s:URLWP>
                </s:PLN>
                <s:SHP>Navi
                  <s:URLWP>URLWP</s:URLWP>
                </s:SHP>
               </rdf:Description>
            </s:index>
        </dc:source>
    </rdf:Description>
</rdf:schema>
```

[30] Si veda §2.4

Figura n. 7

Vediamo una descrizione dello schema RDF:

nelle prime due righe troviamo due distinte dichiarazioni del ***Namespace*** (definisce l'ambito di utilizzo di un insieme di elementi XML per non creare ambiguità); esse specificano che i tag preceduti dal prefisso **:rdf** o **:dc** fanno parte del namespace identificato dall' URI:

xmlns:rdf=http://www.w3.org/2001/XMLSchema

che fa riferimento al Namespace RDF

oppure dall'URI:

xmlns:dc="http://purl.org/dc/elements/1.1/"

che si riferisce al Namespace del *Dublin Core*[31].

È stato scelto il Dublin Core perché è uno schema di *metainformazioni* adatte a qualunque tipo di risorsa, ed è considerato il *nucleo* (*core*) delle metainformazioni. "Dublin" invece deriva dal fatto che è stato creato da un gruppo di sviluppatori riuniti nella città di Dublin in Ohio (USA).
Abbiamo visto che i tag descrittivi di RDF sono teoricamente liberi (§ 2.2); ciò può portare ad una sovrapposizione di tag diversi con lo stesso significato. Per evitare questo problema, RDF propone vocabolari (schemi) per la creazione standard di metadati. Uno di questi schemi è stato suggerito dal Dublin Core Metadata Initiative (DCMI), un'organizzazione che si propone di sviluppare uno standard per la creazione di metadati (quindi proprietà) utili alla definizione di risorse elettroniche (libri, immagini, video ecc.).
Essa ha creato un vocabolario standard per la catalogazione delle più comuni risorse sul Web, che quindi è strutturato come una sorta di *tassonomia*, divisa in due gruppi principali: **Simple** e **Qualified**.
Per il nostro progetto ci siamo serviti del primo, composto da 15 categorie di metainformazioni utili per mappare le risorse presenti in rete.

[31] http://dublincore.org/

Esse si dividono in 3 tipi:

Contenuto	**Proprietà intellettuale**	**Istanza**
Title	*Creator*	Date
Subject	Publisher	Format
Description	Contributor	Identifier
Type	Rights	Language
	Source	
	Relation	
	Coverage	

In corsivo le categorie utilizzate nello Schema RDF del nostro progetto.
Associando un valore a ciascun elemento, si ottengono delle coppie attributo-valore che ci permettono di descrivere la risorsa.
È importante sottolineare come il linguaggio standard del Dublin Core sia implementabile in vari modi; tuttavia, lavorando con il Web Semantico la scelta è ricaduta in modo piuttosto naturale su RDF/XML.

Nei bollettini, i valori dei vari attributi dei tag RDF sono estratti direttamente dalla TAG DOC del file CONLL. Essa è della forma:

<doc url=<URL Bollettino> **index=**<Contatore_Bollettini>
day=<NUMERO_GIORNO_progressivo_(ove presente)>
date=<GIORNO_(numero) MESE_(lettera) ANNO_(numero)
firma=<FIRMA_(ove presente)>**/>**

L' **URI del bollettino** è stata scelta arbitrariamente della forma ***numero_data***: due valori ricavati direttamente dai campi *index* e *data* presenti nella TAG DOC. In questo modo ogni bollettino possiede un'identificazione univoca e priva di ambiguità.
Quindi la descrizione della risorsa "bollettino", sarà rappresentata in RDF come:

```
<rdf:Description rdf:about=
"URI bollettino→numero(index)_data">
```

Come sottoelemento abbiamo poi l'URL del bollettino che, come abbiamo visto, tiene traccia del libro (per la IGM) o del sito (per la IIGM) da cui sono stati ricavati i bollettini. Viene identificato con il tag `<source>`, ovvero:

```
<dc:source>URL bollettino</dc:source>
```

che contiene al suo interno, come ulteriori sottoelementi:

il tag `<s:index>`, ovvero il contatore (numero progressivo) dei bollettini

il tag `<dc:creator>` con la firma (ove esistente) del bollettino

In questo modo viene tenuto traccia di tutte le informazioni principali sui bollettini, ed è quindi possibile passare al testo vero e proprio del bollettino.

Al livello inferiore, scendendo nell'albero RDF, troviamo dunque il tag:

```
<rdf:Description rdf:about="NERType_NumeroNE">
```

Che tiene traccia di ciascuna Entità Nominata presente nel corpus tramite una URI, in modo che le entità siano identificate univocamente.

L'URI della NE è della forma **NERType_NumeroNE**, ovvero:

NERType: il tipo di NER (LOC, PER, SHP, MIL, PLN)
NumNE: il numero che identifica univocamente ciascuna NE e che appunto rimane uguale (per quella NE) per tutto il bollettino.

Il NumNe è stato ricavato mediante il seguente codice Python (fa parte del codice RDF.py che verrà analizzato più avanti):

```
try:
   if not id_nes.has_key(NE):
         id_nes[NE] = counter
         counter+=1
except: #righe bianche, o tag doc
    pass
```

mediante il costrutto try except si determina se una certa NE appartenga o meno ad un dizionario (id_nes).
In caso negativo un contatore viene incrementato di una unità e associato a quella determinata NE. In questo modo ogni NE viene inserita nel dizionario una sola volta.
Alla fine si avranno tante coppie chiave: valore, del tipo:

NE: ID_NE

E nel file CONLL basterà associare quella determinata ID ogni volta che si troverà la stessa NE.
Seguono poi gli altri tag che danno informazioni sul tipo di NER; essi sono contrassegnati dal prefisso **s:** (*statement*) e sono in sostanza predicati che associano una proprietà alla risorsa.
Nel nostro caso specifico la risorsa è la NE, identificata dall'URI NERType_NumeroNE visto sopra, e le sue proprietà sono i tag che identificano il NER_Type.

`<s:PER>, <s:LOC>,<s:SHP>, <s:MIL>, <s:PLN>`

I quali a loro volta contengono al loro interno altri tag

`<s:URLWP>`

che tengono traccia della pagina Wikipedia associata a quella particolare NE.
Il tag `<s:LOC>` ne contiene un altro:

`<s:Geo>`

Esso tiene traccia delle **coordinate geografiche** (***latitudine e longitudine***), estratte dal file luoghi.csv, in cui è raccolta una selezione di luoghi con le rispettive coordinate (vedi §3.2.1). In questo modo è possibile identificare le varianti toponomastiche, che hanno sostanzialmente le stesse coordinate, e risolvere alcuni casi di ambiguità visti in precedenza, dal momento che Latitudine e Longitudine sono, com'è ovvio, univoche.

Per comprendere meglio vediamo un modello grafico dello Schema RDF:

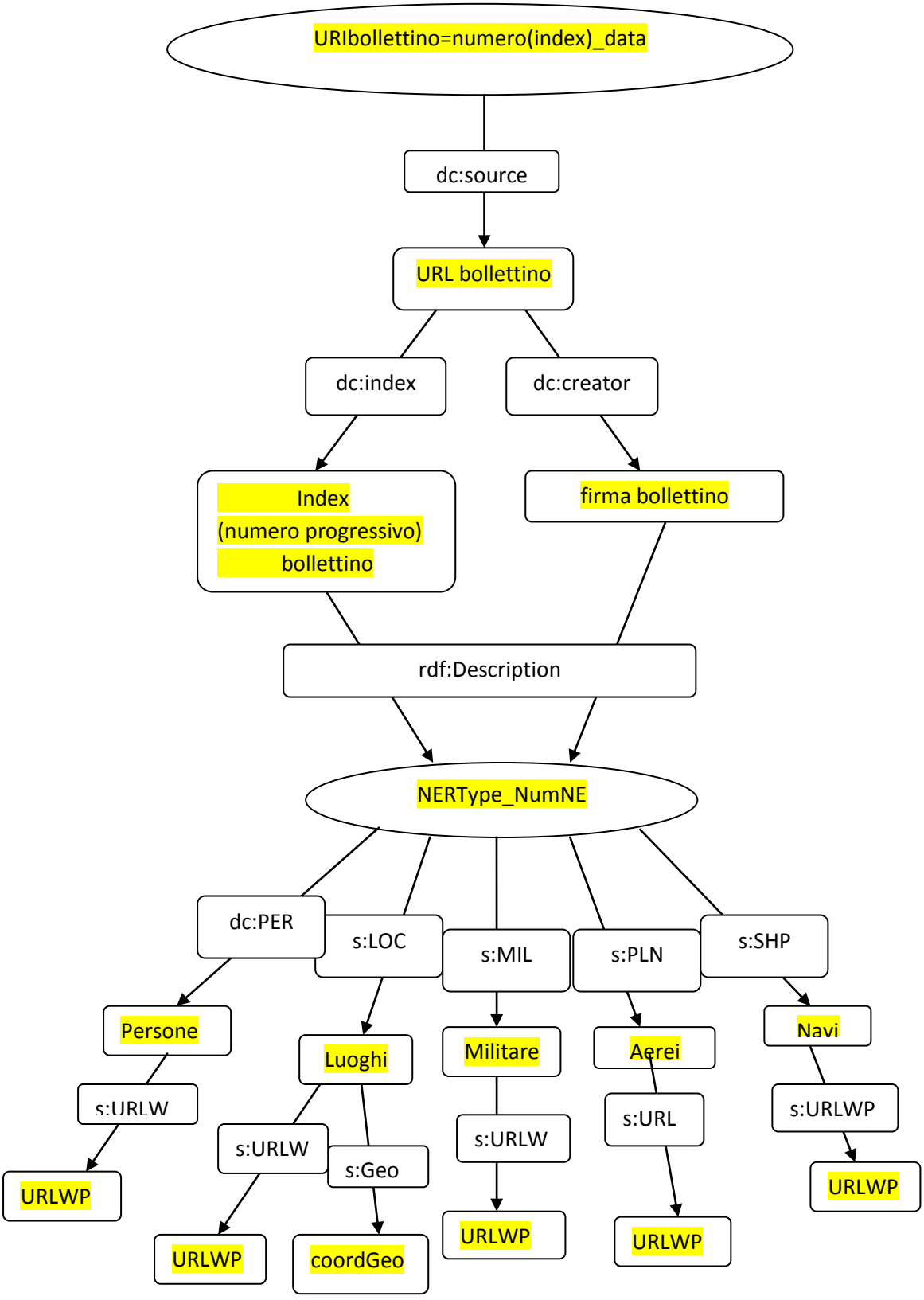

Figura n. 8

3.3.1 Rilascio dei bollettini come Triple RDF

Sulla base dello schema RDF (Figura n.7), si è proceduto alla scrittura di un programma in Python che trasformasse entrambi i corpora di bollettini (I e II GM) in triple RDF. Così facendo si raggiunge l'obiettivo primario di rendere le risorse testuali dei Linked Data.
Il file RDF/XML è generato dallo script RDF.py (vedi appendice codici n.6) che accetta in input il file CONLL esteso con URL Wikipedia e coordinate geografiche (vedi punto 4 del cap. 3.2) e genera in output un file RDF in formato XML.

Vediamo nel dettaglio lo script RDF.py:

All'inizio viene stampato il prologo con la versione di XML, oltre ai namespace RDF e dc:

```
<?xml version="1.0"?>

<rdf:schema xmlns:rdf="http://www.w3.org/2001/XMLSchema"

             xmlns:dc="http://purl.org/dc/elements/1.1/">
```

Successivamente abbiamo la descrizione della risorsa (rdf:Description) con le informazioni generali sul bollettino, recuperate direttamente dal file CONLL "esteso", che contiene tutte le informazioni necessarie (vedi §3.2):

```
<rdf:Description rdf:about="+INDEX+"_"+DATA+”>”+”\n”+\
"\t\t"+"<dc:source>"+URL+"\n"+\
"\t\t\t"+"<dc:index>"+INDEX"+"\n"+\
"\t\t\t"+"<dc:creator>"+FIRMA+"</dc:creator>+"\n"
```

Ricavata direttamente dalla tagdoc:

```
<doc url=<URL Bollettino> index=<Contatore_Bollettini>
day=<NUMERO_GIORNO_progressivo_(ove presente)>
date=<GIORNO_(numero) MESE_(lettera) ANNO_(numero)
firma=<FIRMA_(ove presente)>/>
```

In cui troviamo l'URI del bollettino della forma, scelta arbitrariamente, INDEX_DATA (numero progressivo+data); l'URL del bollettino (il libro o il sito da cui sono stati estratti i bollettini, nel caso rispettivamente della I o II GM).
La FIRMA del bollettino, ove presente.
Per ogni riga viene stampato un numero variabile di *tab* (`\t`) che hanno l'effetto di variare l'indentazione dei tag XML in base alla gerarchia definita nello schema RDF (vedi Figura n. 7).

Dopo di che si entra nel corpo del bollettino:

```
"<rdf:Description rdf:about="+NER_type+"_"+str(nPER)+"\n"+\
"\t\t\t\t\t"+"<dc:PER>"+forma+"\n"+\
"\t\t\t\t\t\t"+"<dc:URLWP>"+URLWP+"</dc:URLWP>"+"\n"+\
"\t\t\t\t\t"+"</dc:PER>"+"\n"+\
"\t\t\t\t"+"</rdf:Description>"
```

In cui abbiamo:
l'URI della NE della forma NER_Type_numero(univoco) della NE;
l'URL Wikipedia (se trovata). A questi si aggiungono le coordinate geografiche nel caso di entità nominate di tipo LOC:

```
"<rdf:Description rdf:about="+NER_type+"_"+str(nLOC"+"\n"+\
"\t\t\t\t\t"+"<dc:LOC>"+forma+"\n"+\
"\t\t\t\t\t\t"+"<dc:Geo>"+LAT+'_'+LONG+"</dc:Geo>"+"\n"+\
"\t\t\t\t\t\t"+"<dc:URLWP>"+URLWP+"</dc:URLWP>"+"\n"+\
"\t\t\t\t\t"+"</dc:LOC"+"\n"+\
"\t\t\t\t"+"</rdf:Description>"
```

4. Stampa dei bollettini

Come ultimo passo tutti i bollettini sono stati stampati in formato testo .txt tramite lo script stampa_corpus.py (vedi Appendice Codici n. 7) senza annotazioni di alcun tipo (né HTML, né XML/RDF) in un numero di file pari al numero di bollettini (1361 per la I GM; 1201 per la II GM).
Ogni file ha per nome *numero* (index) e *data* del bollettino (estratti dalla tagdoc) che, come abbiamo visto, sono identificatori univoci.

```
outfile_name= 'Bollettino#'+ index+'_'+date+'.txt'
```

Mediante un blocco condizionale vengono determinati l'inizio o la fine del singolo bollettino, segnalati rispettivamente dai tag: `<doc>` o `</doc>`.

```
if line.startswith("<doc"):
```

In presenza del tag `<doc>` viene aperto in lettura il singolo bollettino:
`open(os.path.join(output_dir, outfile_name), "wb")`

Tagdoc e nome del file vengono poi inseriti in un file di mapping:
`mapping_file.write (<tagdoc>+ "\t" + outfile_name + "\n")`
In modo da associare il nome di quel determinato bollettino alla sua tagdoc corrispondente.

Il testo dei bollettini è stato ricavato dal file CONLL, precisamente dal suo secondo elemento, cioè la forma:

```
testo= spl_line[1]
```

concatenando ad essa la punteggiatura (identificata nella quarta colonna del CONLL con la lettera F), quando presente:

```
if line.split('\t')[3] == "F":
```

In tal modo si è sopperito anche ad uno dei problemi affrontati all'inizio del progetto,

ovvero quello di non avere tutti i bollettini in formato elettronico: esso può essere il punto di partenza per diverse operazioni di analisi.

5. Conclusioni

I bollettini di guerra, prodotti per esigenze informative e di propaganda durante le Guerre Mondiali, sono adesso disponibili in formato elettronico come triple RDF e sono adatti per svariati tipi di elaborazione, analisi o studi incrociati con risorse affini.

È da sottolineare il fatto, innovativo, che semplici testi cartacei non siano stati semplicemente digitalizzati, ma siano adesso disponibili in un formato che permette di accedere alla loro struttura e di fare interrogazioni anche complesse, cosa che un semplice documento in formato elettronico, ma non strutturato, non permetterebbe.

È ad esempio possibile approfondire le tematiche trattate tramite link a Wikipedia di tutte le parole chiave presenti nel testo. I luoghi, essendo stati georeferenziati non si prestano ad ambiguità di sorta, anche in presenza di omonimie.

Fare ricerche per autore, per data o per luogo diventa ora particolarmente semplice.

Il corpus analizzato è quindi diventato a tutti gli effetti un Web semantico "in scala ridotta" le cui conseguenze sono numerose e tangibili e i cui metodi possono essere applicati a progetti più ampi e su scala maggiore, come ci auguriamo accada in un futuro prossimo.

6. Appendice Codici

1. open_file_Conll.py

(I= CONLL; O= Tutte le line contenenti una NE)

```
#!/usr/bin/python -S
#coding=<utf8>

import sys
import getopt

def read_conll_input(input_file,output_dir):
      outputfilename = output_dir + "\entities.txt"
      with open(outputfilename,"wb") as outfile:
        with open(input_file,"rb") as infile:
            for line_idx, line in enumerate(infile, 1):
                 if line[0].isdigit():
                     uLine = unicode(line.strip(),"utf8")
                     splitted_line = uLine.split("\t")
                     [token_id, forma, lemma] = splitted_line[0:3]
                      ner = splitted_line[-1]
                      if ner != "O":
                          outstring ="\t". join(splitted_line[0:3])
                          outfile.write(outstring.encode("utf8"))
                          outfile.write('\n')
                          print line_idx, uLine
      return
input_file = <path_input>\<CONLL>
output_dir = <path_output>

read_conll_input(input_file,output_dir)
```

2. Extract_entities.py

(I= CONLL; O= NE concatenate)

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import os
import sys
import getopt
import string
from operator import itemgetter
import re

ner_lemma_s = ''
with open (<path_input>\<CONLL>, 'rb') as infile:
  with open(<path_output>\<File_Entity>, 'wb') as
indexes_entity_file:
     for idx, line in enumerate(infile,0):
       if line[0].isdigit():
        line = unicode(line, "utf8").strip()
        spl_line = line.split('\t')
        ner = spl_line[-2]
        if ner.startswith("B-"):
          ner_type = ner.split("-")[1]
          ner_lemma_s = spl_line[1].lstrip("-").capitalize()
#       elif ner.startswith("I-"):
         if spl_line[1]!="di" and spl_line[1]!= "del" and
spl_line[1]!= "dello" and spl_line[1]!= "della" and spl_line[1]!=
"degli" and spl_line[1]!= "dell'" and spl_line[1]!= "d'" and
spl_line[1]!= "in" :
           ner_lemma_s += "_" + spl_line[1].lstrip("-").capitalize()

           else:
             ner_lemma_s += "_" + spl_line[1].lstrip("-")
        else:
           if ner_lemma_s != '':
             outindexes = " ".join([str(idx),ner_type, ner_lemma_s])
             indexes_entity_file.write(outindexes.encode("utf8"))
             indexes_entity_file.write("\n")
             ner_lemma_s = ''
```

3. Stampaurl.py

(I= File_Entities; O= Valuta se esistono le URL WP e le stampa in un FILE con line_id, NER, NE, URL)

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import os
import sys
import getopt
import string
from operator import itemgetter
import re
import urllib2
import glob

found = 0
not_found = 0
wiki_cost = "http://it.wikipedia.org/wiki/"

with open (<path_input>\<File_Entities>, 'rb') as infile:
    with open (<path_output>\<FileUrlEnt>, 'wb') as indirizzi:
        for line in infile:
            uLine = unicode(line.strip(),"utf8")
            spl_line = line.split(' ')
            line_idx = spl_line[0]
            NER=spl_line[1]
            lemma=spl_line[2]
            search_str = "_".join(spl_line[2:])
            if "(" in search_str:
              spl_ps = search_str.split("(")
              search_str = spl_ps[1].strip("_")
            search_str = search_str.strip()
            test_url = wiki_cost + search_str
            test_url = test_url.strip()
            try:
                response = urllib2.urlopen(test_url)
                found+=1
                ind = "\t".join([line_idx,search_str,test_url])
                indirizzi.write(ind.encode("utf8"))
                indirizzi.write("\n")
            except urllib2.HTTPError, err:
                if err.code == 404:
                    not_found+=1
                    ind = "\t".join([line_idx,search_str,"O"])
                    indirizzi.write(ind.encode("utf8"))
                    indirizzi.write("\n")
```

4a. contaEntità.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import os
import sys
import getopt
import string
from operator import itemgetter
import re
import urllib2
import glob

with open('<path_input>\<File_Entities>',"rb") as infile:

    countEnt=0
    countLoc=0
    countMil=0
    countPer=0
    countShp=0
    countPnl=0

    for line in infile:
        uLine = unicode(line.strip(),"utf8")
        splitted_line = line.split(" ")
        if splitted_line[1]!=' ':
            countEnt+=1
        if splitted_line[1]=="LOC":
              countLoc+=1
        elif splitted_line[1]=="MIL":
              countMil+=1
        elif splitted_line[1]=="PER":
              countPer+=1
        elif splitted_line[1]=="SHP":
              countShp+=1
        elif splitted_line[1]=="PLN":
              countPnl+=1
    print
countEnt,"entità","\n",countLoc,"LOC","\n",countMil,"MIL","\n",count
Per,"PER","\n",countShp,"SHP","\n",countPnl,"PNL"
```

4b. contaTokens.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import os
import sys
import getopt
import string
from operator import itemgetter
import re
import urllib2
import glob

with open(<path_input>\<CONLL>, rb") as infile:

    count=0
    for line in infile:

            uLine = unicode(line.strip(),"utf8")
            splitted_line = uLine.split("\t")
            if splitted_line[0].isdigit():
                 count+=1
    print count,"tokens"
```

4c. contaUrl.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import os
import sys
import getopt
import string
from operator import itemgetter
import re
import urllib2
import glob

with open(<path_input>\<FileUrlEnt>,"rb") as infile:

    urlEnt=0
    urlLoc=0
    urlMil=0
    urlPer=0
    urlShp=0
    urlPnl=0
    for line in infile:
        uLine = unicode(line.strip(),"utf8")
        splitted_line = uLine.split("\t")
        if splitted_line[2] !='O':
            urlEnt+=1
            if splitted_line[1]=="LOC":
               urlLoc+=1
            elif splitted_line[1]=="MIL":
               urlMil+=1
            elif splitted_line[1]=="PER":
               urlPer+=1
            elif splitted_line[1]=="SHP":
```

```
                urlShp+=1
            elif splitted_line[1]=="PLN":
                urlPnl+=1

    print
urlEnt,"entità","\n",urlLoc,"LOC","\n",urlMil,"MIL","\n",urlPer,"PER
","\n",urlShp,"SHP","\n",urlPnl,"PNL"
```

5. conll_urls.py

(I=File_UrlEnt, File_CONLL, File_CSV_CoordGEO
O=CONLL_URL_COORDGEO)

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import os
import sys
import getopt
import string
from operator import itemgetter
import re
import urllib2
import glob
import datetime
import csv

def coordinate(coord_file, extended_conll):
      outfile = open(<path_output>\<CONLL_URLWP_CoordGEO>, "wb")

      with open (coord_file, 'rb') as csvfile:
          dict_coord = dict()
          csvreader=csv.reader(csvfile, delimiter=',',
quotechar='"')
          next(csvfile)
          for raw in csvreader:
              uLine = [unicode(csvfield, "utf8") for csvfield in
raw]
              line_idx,guerra,LOC_Name1,LOC_Name2,LOC_Type, LAT,
LONG,_ = uLine
              luoghi=[LOC_Name1.replace(' ', '_'),
LOC_Name2.replace(' ', '_')]
              for luogo in luoghi:
                   dict_coord[luogo] = [LAT,LONG]

      with open (extended_conll, 'rb') as infile2:
          for line_idx, line in enumerate(infile2,1):
              uLine = unicode(line.strip(),"utf8")
              if line[0].isdigit():
                    spl_line=line.split('\t')
                    NER_forma_s=spl_line[-2]
                    URLWP=spl_line[-1]
                    if dict_coord.has_key(NER_forma_s):
                        outstring =  line.strip()+"\t"+ "\t".join([a
for a in dict_coord[NER_forma_s]])
                    else:
```

```
                        outstring =  line.strip()+"\t"+ "Null\tNull"
                else:
                      outstring =  line.strip()

                outfile.write(outstring)
                outfile.write("\n")

def urls2conll(ent_ind_file, conll_file):
      rel_path = os.getcwd()
      filebasename =  os.path.splitext(conll_file)[0]
      outfile_errors = os.path.join(rel_path, filebasename +
'.ERRORS')
      outfile_conll_urls = os.path.join(rel_path, filebasename +
'.ner.doc.urls')
      conll_file = os.path.join(rel_path, conll_file)
      ent_ind_file = os.path.join(rel_path, ent_ind_file)

      print "[Input] CONLL file\t"  + rel_path + conll_file
      print "[Input] Mapping file\t"  + rel_path  + "/"
+ent_ind_file
      print "[Output] Errors in file\t" + outfile_errors
      print "[Output] Urls in conll file\t" + outfile_conll_urls

      idx_dict = dict()
      with open (ent_ind_file, 'rb') as idx_ent:
             with open (conll_file, 'rb') as conll:
                  with open(outfile_conll_urls, "wb") as outfile:
                        with open(outfile_errors, "wb") as
errorsoufile:
                              for line in idx_ent:

                                    line = unicode(line, "utf8")
                                    splitted_line =
line.strip("\n").split("\t")
                                    entity_len =
len(splitted_line[1].split("_"))
                                    counter = entity_len
                                    url = splitted_line[-1]
                                    idx_list = int(splitted_line[0])
                                    tmp_idx = idx_list -1
                                    while (counter>0):
                                          counter-=1
                                          tmp_idx-=1
                                          idx_dict[tmp_idx] =
[splitted_line[1],url]
                               for idx, line in enumerate(conll, -1):
                                    if line.startswith("<"):
outfile.write(line.encode("utf8"))
                                    elif not line.strip():
outfile.write("\n".encode("utf8"))
                                    else:
                                          line = unicode(line,
"utf8").strip()
                                          if line[0].isdigit():
                                                if
idx_dict.has_key(idx):
                                                      substring =
"\t".join(idx_dict[idx])
                                                      outstring =
line.strip() +  "\t" + substring + "\n"
                                                else:
```

```
                                                      outstring =
line.strip() +  "\t" + "O" + "\t" + "O"+ "\n"
                                                      splitted_line
= line.split("\t")
                                                      ner =
splitted_line[-2]
                                                      if ner != "O":

outerrstring = line +  "\tURL NOT FOUND IN MAPPING FILE" + "\n"

errorsoufile.write(outerrstring.encode("utf8"))
                                                  try:
outfile.write(outstring.encode("utf8"))
                                                  except:
                                                        print
outstring
      return outfile_conll_urls
def main(argv):
      print "started : " + str(datetime.datetime.now())
      ent_ind_file = '<File_UrlEnt>'
      conll_file = <File_CONLL>
      outfile_conll_urls = urls2conll(ent_ind_file, conll_file)
      coord_file = <path_input>\<File_CSV_CoordGEO>
coordinate(coord_file, outfile_conll_urls)
      print "finished :" + str(datetime.datetime.now())
if __name__ == '__main__':
    main(sys.argv)
```

6. RDF.py

(I= CONLL_URL_COORDGEO; O= Bollettini in RDF Triple)

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
##
import os
import sys
import getopt
import string
from operator import itemgetter
import re
import urllib2
import glob

NER_forma_s=''
nLOC=0
nPER=0
nMIL=0
nSHP=0
nPLN=0

with open (<Path_Input>\<Conll_URL_CoordGEO>) as CONLL:
         with open (<Path_Output>\<Bollettini_RDF>) as outfile:
              tag1= "<?xml version="+"'1.0'?>"+"\n"+\
```

```
                    "<rdf:RDF
xmlns:rdf="+"'http://www.w3.org/1999/02/22-rdf-syntax-ns#'"+"\n"+\
"\t\t"+"xmlns:dc="+"'http://purl.org/dc/elements/1.1/'"+">"+"<!--
namespace rdf e namespace basato sul Dublin Core-->"
               output = " ".join([tag1])
               outfile.write(output.encode("utf8"))
               id_nes = dict()
               counter = 0
               punt=False
               virg=False
               for line_idx, line in enumerate(CONLL,1):
                   if line[0].startswith('<'):
                     uLine = unicode(line.strip(),"utf8")
                     if line[1:2]!='/':
                       spl_line=uLine.split('"')
                       URL=spl_line[1]
                       FIRMA=spl_line[-2]
                       INDEX=spl_line[3]
                       if INDEX!='0':
                         DATA=spl_line[7].split(' ')
                         if DATA[1]=='gennaio':
                              DATA[1]='01'
                         if DATA[1]=='febbraio':
                              DATA[1]='02'
                         if DATA[1]=='marzo':
                              DATA[1]='03'
                         if DATA[1]=='aprile':
                              DATA[1]='04'
                         if DATA[1]=='maggio':
                              DATA[1]='05'
                         if DATA[1]=='giugno':
                              DATA[1]='06'
                         if DATA[1]=='luglio':
                              DATA[1]='07'
                         if DATA[1]=='agosto':
                              DATA[1]='08'
                         if DATA[1]=='settembre':
                              DATA[1]='09'
                         if DATA[1]=='ottobre':
                              DATA[1]='10'
                         if DATA[1]=='novembre':
                              DATA[1]='11'
                         if DATA[1]=='dicembre':
                              DATA[1]='12'

                         tag2 ="\t"+"<rdf:Description
rdf:about=\""+INDEX+'_'+DATA[0]+'/'+DATA[1]+'/'+DATA[2]+"\">"+"<!--
URI del bollettino della forma numero_data (GG/MM/AAAA)-->"+"\n"+\
                              "\t\t"+"<dc:source>"+URL+"<!--URL del
bollettino tratto dalla TAG DOC oppure il libro da cui sono stati
tratti i bollettini -->"+"\n"+\
                              "\t\t\t"+"<dc:index>"+INDEX+"<!--Indice
che individua univocamente il bollettino tratto dalla TAG DOC--
>"+"\n"+\
"\t\t\t"+"<dc:creator>"+FIRMA+"</dc:creator><!--Firma del bollettino
(dove presente)-->"+"\n"
                       else:
                           tag2 ="\t"+"<rdf:Description
rdf:about=\""+INDEX+"\"><!--URI del bollettino-->"+"\n"+\
                              "\t\t"+"<dc:source>"+URL+"<!--URL del
bollettino tratto dalla TAG DOC oppure il libro da cui sono stati
tratti i bollettini -->"+"\n"+\
```

```
                              "\t\t\t"+"<dc:index>"+INDEX+"<!--Indice
che individua univocamente il bollettino tratto dalla TAG DOC--
>"+"\n"+\
"\t\t\t"+"<dc:creator>"+FIRMA+"</dc:creator><!--Firma del bollettino
(dove presente)-->"+"\n"
                        output = " ".join([tag2])
                        outfile.write(output.encode("utf8"))
                      else:
                         tag3= "\t\t\t"+"</dc:index>"+"\n"+\
                               "\t\t"+ "</dc:source>"+"\n"+\
                               "\t"+ "</rdf:Description>"+"\n"
                         output = " ".join([tag3])
                         outfile.write(output.encode("utf8"))
                    if line[0].isdigit():
                          spl_line = line.split('\t')
                          NER=spl_line[8]
                          NE=spl_line[-4]
                          testo=spl_line[1]
                          URLWP=spl_line[-3]
                          LAT=spl_line[-2]
                          LONG=spl_line[-1]
                          entity_len = len(NE.split("_"))
                          count = entity_len

                          if NER=='O':
                              if line.split('\t')[3] == "F":
                                      if spl_line[1]=='«' or
spl_line[1]=='(' :
                                          punt=True
                                          testo=' '+spl_line[1]
                                      else:
                                        if spl_line[1]=='"' and
virg==False:
                                            testo=' '+spl_line[1]
                                            virg=True
                                            punt=True
                                        elif spl_line[1]=='"' and
virg==True:
                                            testo=spl_line[1]
                                            virg=False
                                            punt=False
                                        else:
                                          punt=False
                                          testo= spl_line[1]
                              else:
                                      if spl_line[1].endswith('-'):
                                          testo='
'+spl_line[1].strip('-')
                                      elif spl_line [7]=='clit':
                                              testo= spl_line[1]
                                      else:
                                              if punt==True:
                                                  testo= spl_line[1]
                                                  punt=False
                                              else:
                                                  testo= ' ' +
spl_line[1]
                              tag4="\t\t\t\t"+"<rdf:Description
rdf:about=Testo>"+testo+"<!--Testo con NER=='O'-->"+"\n"+\
                                     "\t\t\t\t"+"</rdf:Description>"
                              output=" ".join(testo.split('\t'))
                              outfile.write(output)
```

```
                        if  NER!='O':
                                NER_type=NER.split('-')[1]
                                if not id_nes.has_key(NE):
                                        id_nes[NE] = counter
                                        counter+=1
                                for ne, id_ne in id_nes.iteritems():
                                        if ne==NE:
                                               if NER.split('-')[0]
==='B':
                                                      if
NER_type=="LOC":
                                                          tag4=
"\t\t\t\t"+"<rdf:Description
rdf:about=\""+NER_type+"_"+str(id_ne)+"\"><!--URI della NE della
forma NER_type_NumProgressivo-->"+"\n"+\
"\t\t\t\t\t"+"<dc:"+NER_type+">"+NE+"\n"+\
"\t\t\t\t\t\t"+"<dc:Geo>"+LAT+'_'+LONG+"</dc:Geo>"+"\n"+\
"\t\t\t\t\t\t"+"<dc:URLWP>"+URLWP+"</dc:URLWP>"+"\n"+\
"\t\t\t\t\t"+"</dc:"+NER_type+"><!-- Entity di tipo LOC presenti con
attributi per le coordinate geografiche ed eventuali url wikipedia--
>"+"\n"+\
"\t\t\t\t"+"</rdf:Description>"
                                                          output = "
".join([tag4])
outfile.write(output.encode("utf8"))

NER_forma_s=''
                                                      else:
                                                           tag4=
"\t\t\t\t"+"<rdf:Description
rdf:about=\""+NER_type+"_"+str(id_ne)+"\"><!--URI della NE della
forma NER_type_NumProgressivo-->"+"\n"+\
"\t\t\t\t\t"+"<dc:"+NER_type+">"+NE+"\n"+\
"\t\t\t\t\t\t"+"<dc:URLWP>"+URLWP+"</dc:URLWP>"+"\n"+\
"\t\t\t\t\t"+"</dc:"+NER_type+"><!-- Entity di tipo PER presenti ed
eventuali url wikipedia-->"+"\n"+\
"\t\t\t\t"+"</rdf:Description>"
                                                           output = "
".join([tag4])
outfile.write(output.encode("utf8"))
NER_forma_s=''
                                                  else:
                                                      pass
                tag5="</rdf:RDF>"
                output = " ".join([tag5])
                outfile.write(output.encode("utf8"))
```

7. Stampa_corpus.py
(I=File CONLL; O=Bollettini.txt)

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import os
import re
import sys
import getopt
from collections import defaultdict


def usage():
    print """
General Usage:
> python weedsPrec.py [-i input]

    [-h|--help]     >> print this help
    [-i|--input]    >> path/to/input/file
"""

def split_corpus(input_file, output_dir):
    input_dir = os.path.dirname(os.path.realpath(input_file))

    mapping_file = open(os.path.join(input_dir, "mapping_file"),
"wb")
    smallfile_prefix = "BOLLETTINO #"
    outfile_name = ''
    with open(input_file, mode="rb") as bigfile:
        file_count = 0
        opened_file = False
        punt=False
        virg=False
        for line in bigfile:
            if line.startswith("\t_"):
                pass
            elif line.startswith("<doc"):
                doc_url = line.strip()
                spl_line = line.split('"')
                index=spl_line[3].strip('"')
                if index!='0':
                        date=spl_line[5].strip('"')
                        outfile_name= smallfile_prefix+
index+'_'+date+'.txt'
                        file_count += 1
                        smallfile = open(os.path.join(output_dir,
outfile_name), "wb")
                        opened_file = True
                        mapping_file.write(doc_url+ "\t" +
outfile_name + "\n")
                else:
                         outfile_name= smallfile_prefix +
index+'.txt'
```

```
                        file_count += 1
                        smallfile = open(os.path.join(output_dir,
outfile_name), "wb")
                        opened_file = True
                        mapping_file.write(doc_url+ "\t" +
outfile_name + "\n")

            elif line.startswith("</doc>"):
                smallfile.close()
                opened_file = False
            elif opened_file is True:
                if line.strip():
                    spl_line = line.split('\t')
                    if line.split('\t')[3] == "F":
                        if spl_line[1]=='«' or spl_line[1]=='(' :
                            punt=True
                            testo=' '+spl_line[1]
                        else:
                          if spl_line[1]=='"' and virg==False:
                              testo=' '+spl_line[1]
                              virg=True
                              punt=True
                          elif spl_line[1]=='"' and virg==True:
                              testo=spl_line[1]
                              virg=False
                              punt=False
                          else:
                            punt=False
                            testo= spl_line[1]
                    else:
                        if spl_line[1].endswith('-'):
                            testo=' '+spl_line[1].strip('-')
                        elif spl_line [7]=='clit':
                                testo= spl_line[1]
                        else:
                                if punt==True:
                                    testo= spl_line[1]
                                    punt=False
                                else:
                                       testo= ' ' + spl_line[1]
                    smallfile.write(testo)
    mapping_file.close()
input_file = <Path_Input>\<Conll>
output_dir = <Path_Output>\<OutputDirectory>
print 'Input file :', input_file
print 'Output directory :', output_dir

split_corpus(input_file, output_dir)
```

7. Bibliografia e Sitografia

Analisi computazionale dei bollettini (CoLing Lab)

Boschetti, F., Cimino, A., Dell'Orletta, F., Lebani, G.E., Passaro, L., Picchi, P., Venturi, G., Montemagni, S., Lenci, A., (2014), *Computational Analysis of Historical Documents: An Application to Italian War Bulletins in World War I and II*, Pisa

Passaro, L., Lenci, A., (2014), *Il Piave mormorava…": Recognizing Locations and other Named Entities in Italian Texts on the Great War*.
In *Proceedings of the First Italian Conference on Computational Linguistics CLiC-it 2014 & the Fourth International Workshop EVALITA 2014*, pp. 286-290

Ide, N., Woolner, D., (2004), Exploiting *Semantic Web Technologies for Intelligent Access to Historical Documents*.
In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*, Lisbona, Portogallo, pp. 2177-2180

L'interfaccia grafica del progetto del CoLing Lab http://www.memoriediguerra.it/ [pagina consultata il 21/01/2015]

Bollettini

I Bollettini della Guerra 1915-1918, (1923), prefazione di Benito Mussolini, Milano, Alpes

Bollettini di guerra del Comando Supremo: 1940-1943, (1970), Stato maggiore dell'Esercito, Ufficio Storico, Roma

Bollettini della II GM in formato PDF:
http://www.regioesercito.it/bollettini/indexboll.htm
[pagina consultata il 20/01/2015]

Bollettini della seconda Guerra Mondiale, a cura di Roberto Stocchetti, (2009), http://www.alieuomini.it/pagine/dettaglio/bollettini_di_guerra
[pagina consultata il 19/01/2015]

Cronache della guerra, annate 1940-1943, Tuminelli & C., Roma

Linked Data

Berners Lee, T., (2006, agg. 2009), *Linked Data*,
http://www.w3.org/DesignIssues/LinkedData.html
[pagina consultata il 20/01/2015]

Bizer, C., Heath, T. and Berners-Lee, T. (2009). "Linked Data - the story so far", *International Journal on Semantic Web and Information Systems*, 5, (3), 1-22
Disponibile anche in formato pdf a:
http://tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf
[pagina consultata il 05/02/2015]

Database di entità geografiche http://www.geonames.org/
[pagina consultata il 20/01/2015]

DBPedia, http://dbpedia.org/About
[pagina consultata il 20/01/2015]

DBPedia (versione italiana), http://it.dbpedia.org/
[pagina consultata il 20/01/2015]

DCMI (Dublin Core Metadata Initiative), http://dublincore.org/
[pagina consultata il 08/02/2015]

Extensible Markup Language (XML), (Agg. 26/01/2015), http://www.w3.org/XML/
[pagina consultata il 08/02/2015]

Gandon, F., Schreiber, G., (Agg. 25/02/2014), RDF 1.1 XML Syntax,
http://www.w3.org/TR/rdf-syntax-grammar/#nodeElementList
[pagina consultata il 09/02/2015]

Guerrini, M., Possemato, T., "Linked data: un nuovo alfabeto del web semantico", *Biblioteche oggi*, aprile 2012: 7-15
Disponibile anche in formato pdf a:
http://www.bibliotecheoggi.it/content/201200300701.pdf
[pagina consultata il 04/02/2015]

Heath, T., Bizer, C., (2011), *Linked Data: Evolving the Web into a Global Data Space. Synthesis Lectures on the Semantic Web: Theory and Technology* 2°ed., Morgan & Claypool,
disponibile anche in formato html a: http://linkeddatabook.com/editions/1.0/
[pagina consultata il 04/02/2015]

Linked Data - Connect Distributed Data across the Web, http://linkeddata.org/
[pagina consultata il 20/01/2015]

Linking Open Data. W3C SWEO Community Project,
http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData
[pagina consultata il 20/01/2015]

Ontologie

Gruber, T. R., (1993), "A Translation Approach to Portable Ontology Specifications".
In *Knowledge Acquisition*, 5(2), (1993): 199-220 disponibile anche in formato PDF:
http://tomgruber.org/writing/ontolingua-kaj-1993.pdf

Piattaforma per l'e-learning basata su strutture ontologie,
http://www.merlot.org/merlot/index.htm
[pagina consultata il 20/01/2015]

K. Smith, M., Welty C., L. Mc Guinness D., (2004), *OWL Web Ontology Language Guide*, http://www.w3.org/TR/2004/REC-owl-guide20040210/#owl_equivalentClass (agg. 2009) http://www.w3.org/TR/owl-guide/
[pagina consultata il 10/02/2015]

URI

Berners-Lee, T., et. al., (2005), *Uniform Resource Identifiers (URI): Generic Syntax*,
http://www.ietf.org/rfc/rfc3986.txt
[pagina consultata il 09/02/2015]

Sauermann, L., Cyganiak, R., (2008), *Cool URIs for the Semantic Web*,
http://www.w3.org/TR/2008/NOTE-cooluris-20080331/
[pagina consultata il 20/01/2015]

Web Semantico

Berners-Lee T., (1998), *What the Semantic Web can represent*,
http://www.w3.org/DesignIssues/RDFnot.html
[pagina consultata il 21/01/2015]

Berners Lee, T., (Settembre 1998 - agg. Ottobre 1998), *Semantic Web Road map. An attempt to give a high-level plan of the architecture of the Semantic WWW*,
http://www.w3.org/DesignIssues/Semantic.html
[pagina consultata il 19/01/2015]

Berners-Lee, T**.**, Connolly, D., Swick, R. R., (1999), *Web Architecture: Describing and Exchanging Data***,** http://www.w3.org/1999/04/WebData
[pagina consultata il 20/01/2015]

Berners Lee, T., Fischetti, M., (1999), *Weaving the Web*. *The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*, Harper, San Francisco

Berners-Lee, T., (2000), *Semantic Web - XML2000*,
http://www.w3.org/2000/Talks/1206-xml2k-tbl/Overview.html
[pagina consultata il 20/01/2015]

Berners-Lee, T., (2001), *L'architettura del nuovo Web. Dall'Inventore della Rete il progetto di una comunicazione democratica*, Tr. it. G. Carlotti, Feltrinelli

Berners-Lee, T., Hendler, J., Lassila, O., (2001), "The Semantic Web". In *Scientific American*. Disponibile anche in formato HTML:
http://www.cs.umd.edu/~golbeck/LBSC690/SemanticWeb.html
[pagina consultata il 19/01/2015]

Dorati, A., Costantini, S., *Approcci al Web Semantico*,
http://www.websemantico.org/articoli/approcciwebsemantico.php
[pagina consultata il 09/02/2015]

Hawke, S., (2001), *How We Identify Things (on the Semantic Web)*?,
http://www.w3.org/2001/03/identification-problem/
[pagina consultata il 20/01/2015]

Swartz, A., (2002), *The Semantic Web In Breadth*,
http://logicerror.com/semanticWeb-long
[pagina consultata il 20/01/2015]

Swartz, A., (2002), *The Semantic Web (for Web Developers)*,
http://logicerror.com/semanticWeb-webdev
[pagina consultata il 20/01/2015]

Gallinaro M., *Il Web Semantico*, (2010), Dispense tratte dalla tesi di laurea di Natalino Fiacco, Corso di Comunicazione d'impresa, Università La Sapienza, Roma,
http://www.slideshare.net/stefanoepifani/dispensa-5-il-web-semantico?src=embed
[pagina consultata il 08/02/2015]