**Information Extraction per la geolocalizzazione con GATE/JAPE**

**Corso di laurea triennale in ingegneria informatica**

**Università degli Studi di Firenze**

*Relatore: Tenti Marco*

tentimarco0@gmail.com,tentim@infinito.it

*Correlatori: Gianni Pantaleo*

08/10/2014

***Sommario:***

*A partire da un database di URL ,si descrive un sistema per la ricerca ed estrapolazione di espressioni chiave dalle pagine web (HTML,PDF,ecc.), per la geolocalizzazione dei soggetti descritti dal contenuto delle pagine , con l’ausilio del software GATE /JAPE.*

**Indice**

1. **[Introduzione .………………………………………………………………….4](#Introduzione)**
2. [Note per semplificare ed evidenziare la comprensione e il contenuto del testo](#Note_personali_sul_contenuto)
3. [Strumenti e software utilizzati](#Strumenti_e_software_utilizzati)
4. [Veloce occhiata alla struttura del sistema OntoStar](#Veloce_occhiata_Ontostar)
5. [Idea iniziale del progetto](#Idea_iniziale_progetto)
6. [**Acquisire le informazioni nel Web …………………………………..6**](#Acquisire_informazione_nel_web)
   * + 1. [Information Retrieval (IR)](#IR)
       2. [Information Extraction (IE)](#IE)
       3. [I problemi delle informazioni non strutturate del web](#I_problemi_delle_informazioni)
7. [**Il web Semantico (Semantic Web Vision) ………………………11**](#Web_Semantico)
   1. [Il modello dei dati nel web](#Modello_dati_web)
   2. [Dati ,Metadati, RDF e simili](#Dati_Metadati_RDF)
   3. [Le annotazioni semantiche](#Semantic_Annotation)
   4. [Le Ontologie](#Ontologie)
   5. [Conclusione sulla Semantica Web](#Conclusione_semantica_web)
8. [**GATE - General Architecture Text Engineering ………………19**](#GATE)
9. [Generazione dei Corpora](#Generazione_corpora)
10. [ANNIE – A Nearly New Information Extraction System](#ANNIE)
    * 1. [Document Reset](#Document_reset)
      2. [Tokeniser](#Tokenizer)
      3. [Sentence Splitter](#Sentence_splitter)
      4. [Part Of Speech Tagger (POS Tagger) e Generic Tagger](#POS_Tagger)
      5. [Gazetteer](#Gazetteer)
      6. [JAPE Transducer e ANNIE NE Transducer](#Transducer)
      7. [Orthomatcher](#Orthomatcher)
11. [Considerazioni sull’uso di GATE](#Considerazioni_GATE)
12. [**Le annotazioni di GATE e HTML ……………………………………30**](#le_annotazioni_di_gate_e_html)
13. [**Con le regole JAPE**](#Con_Jape)
14. [**Con la componente Annotation Set Transfer**](#Con_Annotation_Set_Transfer)
15. [**Con il componente Flexible Gazetter**](#Con_Flexible_Gazetter)
16. [**Identificazione della lingua del documento web ……………36**](#identificazione_della_lingua_documento_w)
17. [Scelta della lingua attraverso il Conditional Corpus](#Scelta_della_lingua_attraverso_il_Condit)
18. [Scelta della lingua con Lingpipe Language Identifier](#Scelta_della_lingua_con_lingpipe_languag)
19. [Scelta della lingua con le Feature del documento HTML](#Scelta_della_lingua_con_le_feature)
20. [Le modalità di scelta della lingua](#modalità_di_scelta_lingua)
21. [Conclusione sulla modalità di scelta della lingua per questo studio](#conclusioni_sulle_modalità_di_scelta_lin)
22. [**Progetto Java: EstrazioneDatiWebPageToRecordSQLWithGATE………………42**](#progetto_java_geolocationwebpagewithgate)
23. [La nostra prima classe JAVA](#Primi_pass_con_GATE_EMBEDDED)
24. [Il file .gapp/.xgapp](#il_file_gapp_xgapp)
25. [Le regole JAPE](#le_regole_jape)
26. [Struttura del progetto](#Struttura_programma_Java_1)
27. [Gli oggetti java del nostro progetto](#Gli_oggetti_java_del_nostro_progetto)
28. [I database del progetto](#I_database_del_progetto)
29. [Costruzione del Corpus degli URL](#Costruzione_del_corpus_degli_url)
30. [Configurazione locale di GATE](#Configurazione_locale_di_gate)
31. [Estrazione e gestione del contenuto delle Annotazioni](#Estrazione_e_gestione_del_contenuto)
32. [Configurazione chiamata/risposta con l’API Google Maps](#Configurazione_chiamata_risposta_con_API)
33. [Integrazione dei dati con Ontostar e inserimento dei GeoDument](#Integrazione_dei_dati_con_ontostar)
34. [Integrazione dei dati con il motore di ricerca Yahoo SearchMonkey](#SearchMonkey)
35. [Integrazione dei dati con Apache Tika](#Integrazione_dei_dati_con_Apache_Tika)
36. [Integrazione delle Regioni e delle Provincie](#Integrazione_delle_regioni_provincie)
37. [Ultima classe java:Upgrade della tabella MySQL per la successiva integrazione con un’Ontologia](#Ultima_classe_java_Upgrade_della_tabella)
38. [Creazione della tabella GeoDomainDocument per la successiva validazione del progetto](#creazione_tabella_geodomaindocument)
39. [Output del programma java e diagramma UML finale](#Output_finale_java)
40. [**Validazione del progetto e Conclusione …………………………72**](#validazione_del_progetto)

**[APPENDICE ………………………………………………………………………………………78](#APPENDICE)**

1. [**Introduzione veloce all’utilizzo dei file .par**](#A_Introduzione_par)
2. [**Link utili alla documentazione di GATE**](#B_Link_utili)
3. [**Programmazione delle regole JAPE**](#C_programmazione_Jape)
4. [**Customizzazione della PIPELINE di GATE**](#D_customizzazione_della_piepline_di_gate)
5. **Introduzione**

Lo studio condotto in questa relazione nasce, a partire dal lavoro precedentemente svolto dagli studenti Giulio Galvan, Tommaso Levato, Alessio Sarullo, che hanno realizzato un sistema di nome Ontostar , tale che , fornito un database di URL (circa 6 milioni nel nostro caso) che puntano alle rispettive pagine web/documenti(HTML,PDF,ecc.), è in grado di estrarre espressioni chiave da tale database di documenti scaricati dal web, il nostro studio permette di integrare a tali espressioni chiave ulteriori espressioni chiave (Keyword) inerenti alla sola posizione geografica a cui fanno riferimento, le informazioni nel contenuto delle pagine web.

In altre parole andiamo a estrarre tutte quelle informazioni che possono servire per la geo localizzazione in particolare tutte quelle informazioni che è possibile utilizzare in un secondo momento per il tracciamento della posizione su una mappa grafica (tipo Google Maps), quindi latitudine, longitudine, via, città, provincia, ecc.

Il nostro progetto si limiterà all'estrazione delle informazioni di geo localizzazione e all’invio di esse al sistema Ontostar (non ci interesseremo al come vengono elaborati questi dati da tale sistema), attraverso l’utilizzo di vari strumenti e software.

1. **Note per semplificare ed evidenziare la comprensione e il contenuto del testo**

Durante la stesura del progetto si sono riscontrati problemi di varia natura, GATE è un ottimo programma ma è Open source e non tiene conto di tutte le variabili che un progettista riscontra durante lo svolgimento del suo lavoro, personalmente sono riuscito a fare apparire l’alert che dice “*Congratulations you found the only bug in GATE*” più di una volta , **oltre alla descrizione del progetto fornisco anche una guida ai primi passi con GATE e le regole JAPE (guardate le APPENDICI fornite) questo per far capire al lettore (alle prime armi con questo genere di cose), di cosa stiamo parlando e di come ci muoviamo di conseguenza nel progetto.**

Inoltre poiché sono stati riscontrati diversi problemi e soluzioni sia di stesura personale e non, nel corso della fase di realizzazione della pipeline di GATE e delle regole JAPE per aiutare il lettore a seguirne lo svolgimento come in molti libri aggiungo delle piccole icone per marcare i punti più importanti:

|  |  |
| --- | --- |
|  | Riflessioni, note e ipotesi fatte durante la realizzazione dello studio. |
|  | Riflessioni e note importanti da ricordare. |
|  | Riflessioni e note dei GATE USER[[1]](#footnote-1) comunità di utenti che discute gli argomenti di programmazione con GATE, JAPE e web service che ne fanno uso |

1. **Strumenti e software utilizzati**

* **Netbeans**, il famoso software IDE per programmare con qualsiasi linguaggio.
  + - **NOTA**: Non esiste un editor o un debug ufficiale per i file con estensione.jape, l’unica è aprirli come file .java
    - **Osservazione**, Netbeans è il migliore ma lavorare con la sintassi jape per un file java fa apparire segnalazioni di errori a raffica, se la cosa vi da noia vi consiglio di scrivere le vostre regole jape con software come **PSpad** o **Notepad++** e solo in fase di gestione delle eccezioni o di scrittura del corpo java del jape ritornare al più efficiente Netbeans.
* **Workbench MySQL,** il famosissimo, editor per la gestione dei database, in sede di progetto verrà utilizzato solo per lo storage dei database.
* **GATE**, il software con editor incorporato per l’”Information Extraction” nei file di testo e la creazione di annotazioni semantiche e molto altro sarà visto meglio più avanti.
* **QuickPar and SmartPar** semplici software per la lettura e scrittura di file con estensione .par.

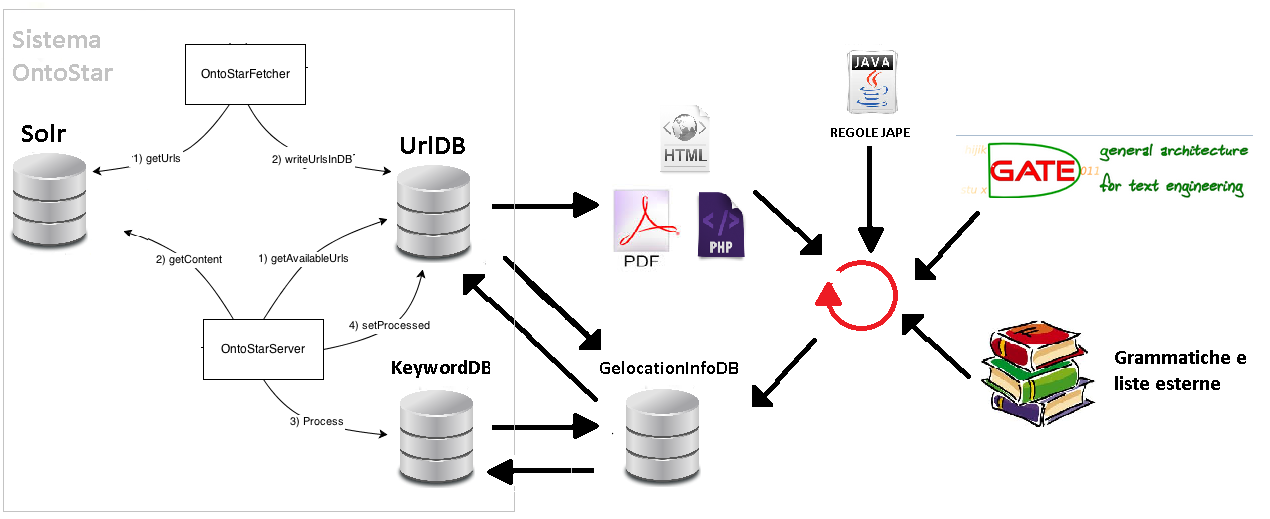
1. **Veloce occhiata alla struttura del sistema Ontostar**

Il sistema ha due elementi principali: uno si occupa del crawling e dello storage delle pagine web(Ontostar suite implementato con OntostarFetcher), l'altro provvede all'estrazione delle parole chiave(OntostarServer).

**Fig.1:Il sistema Ontostar**

1. **Idea iniziale del progetto**



**Fig.2:Idea inziale del nostro progetto e su come si interlaccia con il sistema Ontostar**

1. **Acquisire l’Informazione nel Web**

Per parlare di classificazione delle informazioni è necessario innanzitutto poter accedere ai documenti e “controllare” il loro contenuto. La grande maggioranza di documenti disponibili nei computer e nel web è costituita da testo: per esempio i giornali, i report aziendali, le pagine web, gli articoli scientifici, i documenti medici; il testo può presentarsi in vari formati elettronici (Word, PDF, PS, HTML, XML), in varie lingue (inglese, italiano, giapponese), in vari caratteri di codifica (ASCII,Unicode). Per un elaboratore, il testo è inteso come un insieme di caratteri alfanumerici e speciali; per un utente, il testo rappresenta un insieme di dati che, interpretati attraverso la conoscenza e in un preciso contesto, si trasformano in informazione.

L'accesso tradizionale alle informazioni è rappresentato dall'Information Retrieval (IR) e dall'Information Extraction (IE) e, più recentemente, anche da altri campi di ricerca tra cui Question Answering e Text Understanding.



**Fig.3: Le varie metodologie per l’accesso all' informazione nel documento di testo**

Queste aree, per le quali non esiste una netta separazione, rappresentano alcuni modi con cui possiamo ottenere informazione dai dati testuali. Secondo gli scopi, un documento di testo può essere considerato come una semplice sequenza di simboli codificati (per un computer), oppure come una sequenza di parole con un possibile significato (IR), o ancora come una sequenza di frasi con significato possibilmente rilevanti per un argomento (IE). La successiva sfida è cogliere il senso generale di un documento per poi rispondere a domande sullo stesso (TU) oppure saper elaborare domande poste dall'utente e riuscire a formulare risposte contenenti l'informazione desiderata senza costringere l'utente a setacciare un lungo elenco di documenti (QA). Ottenere informazioni è un processo complesso, sia perché i dati spesso si presentano non strutturati, sia perché le collezioni di documenti diventano sempre più estese (information overload), senza contare che in alcuni casi l'informazione è duplicata o inesatta. La velocità con cui le collezioni s' ingrandiscono, l'eterogeneità e la complessità che le caratterizzano, nonché le richieste sempre più esigenti dell'utente sembrano sminuire, se non vanificare, i costanti miglioramenti introdotti per l'accesso ai documenti.

*Per avere un'idea delle problematiche che nascono dal processo di elaborazione dei documenti, si fornisce una panoramica dei primi due approcci sopra citati: l'IR perché vanta una lunga esperienza in materia di accesso e gestione dei documenti tramite IE perché, oltre a basarsi su numerosi risultati ottenuti dall'IR, sarà l'approccio utilizzato in questo studio.*

* 1. **Information Retrieval (IR)**

IR, una disciplina scientifica con vasta esperienza in materia di accesso e gestione dei documenti.

*“L'Information Retrieval e un campo relativo alla struttura, analisi,* *organizzazione, memorizzazione, ricerca e recupero di informazione (1968)*[[2]](#footnote-2).

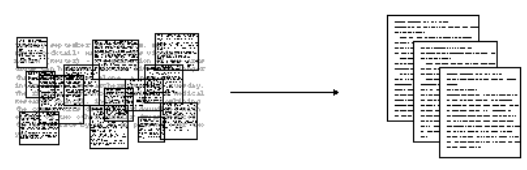
Una definizione alternativa di esattamente quarant’anni dopo parla di:

*“Ricerca di materiale di natura non strutturata presente all'interno di una vasta collezione che soddisfa un bisogno informativo (2008)”[[3]](#footnote-3)*.

Adattando la seconda definizione al nostro caso, potremmo descrivere il nostro studio come una ricerca mirata di referti scritti in testo non strutturato e presenti in un repository, ai fini di una loro classificazione. Il termine “informazione”, infatti, è molto generale e ben si adatta alle esigenze odierne che comprendono, oltre al reperimento d’informazione su documenti di testo, anche quello inerente ai contenuti multimediali come immagini, video e audio (per esempio speech e musica). Focalizzando la nostra attenzione sui documenti di testo, ci accorgiamo subito che, pur avendo qualche accenno di struttura come titolo, autore data o abstract, si presenta con un'informazione non strutturata difficile da descrivere e reperire con un algoritmo, contrariamente a quanto accade ai dati contenuti nei record di un database, rigorosamente strutturati in tabelle e campi, e consultabili con apposite query.

Con la diffusione d'internet e la grande disponibilità di dati, il processo di reperimento dell'informazione è diventato la forma dominante di accesso ai dati stessi, ma sebbene si assista a numerosi sforzi per rendere l'informazione facilmente elaborabile dalle macchine (pensiamo per esempio alle pagine web scritte in XML), molta informazione si presenta ancora non-strutturata, quindi non chiara, non semanticamente evidente e non “facilmente maneggiabile” per un computer, come lo è invece l'informazione memorizzata in un database.

Se pensiamo a una pagina web possiamo eventualmente parlare di informazione semi-strutturata, in quanto sono presenti marcatori per identificare il titolo e qualche altro elemento del corpo, marcatori che ci consentono di trarre “qualche” informazione sul contenuto della pagina web stessa. Tuttavia, tale processo è spesso molto impreciso e inaffidabile, soprattutto quando l'autore della pagina si è preoccupato più dell'aspetto estetico della pagina che non della “strutturazione” dei contenuti.



**Fig. 4 :IR , Information Retrieval**

IR: *ottiene insiemi di documenti pertinenti cioè si analizzano i documenti.*

* 1. **Information Extraction (IE)**

Per “information extraction” s'intende la creazione di una rappresentazione strutturata dell'informazione rilevante presente in un documento machine-readable non strutturato. Se applicata a documenti testuali, l'Information Extraction è dunque una tecnologia che punta a estrarre elementi salienti relativi ad un particolare contesto, come entità o relazioni.

Esso si distingue dall'IR perché l'enfasi in IR è trovare documenti che già contengono la risposta alla domanda formulata dell'utente: data una collezione di documenti, il sistema di IR che riceve in input una query (set di parole chiave) seleziona un sottoinsieme di documenti che ritiene rilevanti per la query. L’utente poi navigherà la lista di documenti e cercherà l’informazione che più gli interessa. Il sistema di IE, invece, data una selezione di documenti, cerca invece di estrarre in modo strutturato l’informazione rilevante secondo le esigenze fornite in input.

IE è usata in numerose applicazioni, in particolare nel text data mining: il data mining estrae sapere o conoscenza a partire da grandi quantità di dati, attraverso metodi automatici o semi-automatici. Il text data mining è una forma particolare di data mining, dove i dati sono costituiti da testi scritti in linguaggio naturale, quindi da documenti "destrutturati". Il text data mining unisce la tecnologia della lingua con gli algoritmi del data mining[[4]](#footnote-4).

IR e IE sono quindi tecnologie complementari che condividono lo stesso obiettivo finale: l'estrazione d'informazione implicita contenuta in un insieme di documenti. Nei motori di ricerca, l'uso primario di IE è l'identificazione di attributi (features) da utilizzare per migliorare il ranking. Con eccessivo ottimismo, si pensa che le tecniche di IE possano trasformare la ricerca su testo in un problema d'interrogazione su database, estraendo tutte le informazioni importanti dal testo e memorizzandole in forma strutturata, ma per ora si tratta di obbiettivi ancora lontani. Esempi di attributi da identificare possono essere: titolo del documento, testo in grassetto, sostantivi. Se il documento è scritto in XML, alcuni di questi dati sono facilmente individuabili tramite i markup; altri dati, invece, richiedono un processo aggiuntivo prima di annotare il testo con la relativa feature (esempio: sentence). Tuttavia, se si considerano documenti acquisiti mediante tecniche di riconoscimento ottico dei caratteri (OCR), essi non hanno alcun markup e, quindi, anche semplici elementi strutturali come il titolo devono essere individuati e annotati.

Sebbene le feature siano spesso di tipo generale, gli sforzi in IE si concentrano per ottenerne di più complesse, con uno specifico contenuto semantico, come entità e relazioni. La named entity è una parola o sequenza di parole utilizzata per riferirsi a qualcosa di particolarmente significativo in un determinato contesto o per una specifica applicazione. Gli esempi più comuni sono i nomi di persona, compagnie, organizzazioni, luoghi, date, quantità, valute monetarie.

Data, la natura più specifica di queste feature, il processo di identificazione e di marcatura è noto come semantic annotation e generalmente gli approcci più usati come identificatori di named entity sono basati su regole che usano uno o più lexicons (liste di parole o gruppi di parole). Se la lista è sufficientemente precisa ed estesa, gran parte dell'estrazione può essere fatta semplicemente tramite lookup; combinando regole e patterns, poi, si possono creare nuove entità (per esempio il pattern via <word>, <number> può identificare una via).

L'IE si compone generalmente di una serie di fasi (task):

* Named Entity Task (NE): marcare nel testo le stringhe che rappresentano entità.
* Template Element Task (TE): estrarre l’informazione rilevante associata alle entità.
* Template Relation Task (TR): estrarre l’informazione sulle relazioni tra gli elementi del Template ottenuti con NE e TE.
* Scenario Template Task (ST): come TR, ma la scelta del template (slot da riempire) è determinata dal riconoscimento di elementi rilevanti per un certo argomento.
* Coreference Task (CO): catturare l’informazione sulle coreferenze legate agli elementi marcati con TE e NE.

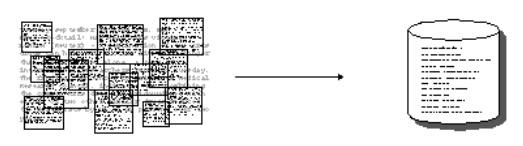
O più semplicemente si può immaginare lo IE come somma di tre fasi:

**IE =(segmentazione + classificazione + associazione)**

Applicandole all'esempio di seguito riportato, si ottengono i risultati evidenziati in tabella:

*“Il direttore dei grandi magazzini Luna, Domenico Zorro, stanco della crisi che abbatte da mesi le vendite, ha promosso un periodo di forti sconti. Da domani, nei negozi Luna, 150 articoli in promozione con sconto 50%. Zorro ha espresso ottimismo nelle vendite. Il vicedirettore dei grandi magazzini Sole, Arturo Batman, ha dichiarato che tali iniziative dovrebbero essere concordate. I magazzini Sole, da sempre concorrenti dei magazzini Luna, temono infatti un improvviso calo delle vendite. Nella vicenda è intervenuto il sindaco, Matteo Spider il quale … “*

|  |  |  |
| --- | --- | --- |
| **Segmentazione** | **Classificazione** | **Associazione** |
| Direttore  Luna  Domenico Zorro  Luna  Zorro  Vicedirettore  Sole  Arturo Batman  …………………………………… | Direttore JOB  Luna NAME  Domenico Zorro PERSON  Luna NAME  Zorro PERSON  Vicedirettore JOB  Sole NAME  Arturo Batman PERSON  ………………………………………………. | Direttore JOB  Luna NAME  Domenico Zorro PERSON  Luna NAME  Zorro PERSON  vicedirettore JOB  ……………………………………………… |



**Fig.5 :IE, Information Extraction**

IE: ottiene fatti su documenti cioè si analizzano i fatti

* 1. **I problemi delle informazioni non strutturate del web**

L'ambiente web ormai rappresenta una piattaforma di scambio per i dati insostituibile. Se prima del suo utilizzo in termini massicci, ogni azienda o struttura organizzata aveva una propria collezione di dati cui accedeva con applicazioni specializzate e dedicate, oggi la necessità di scambio e di condivisione delle informazioni tramite il web ha fatto ben notare la difficoltà di conciliare reperimento d' informazione, utilizzo di banche dati e testo non strutturato. La diffusione della connettività internet ha favorito un uso intensivo della stessa da parte degli utenti accelerando nello stesso tempo la produzione sempre più abbondante che contenuti web. Contestualmente non si è posta sufficiente attenzione alla struttura dei documenti web, pertanto se da un lato i motori di ricerca si sono resi necessari per consultare tali contenuti (e la loro creazione ha decretato il vero e proprio successo del web), dall'altro il loro uso comporta alcuni problemi:

**Alto richiamo, bassa precisione**: La produzione di informazione ha creato “sovrabbondanza” in questa situazione, l'utente che sottopone una richiesta al motore di ricerca, anche se ben formata, si vede restituire un set di documenti che comporta spesso un alto costo in termini di tempo ed energie per selezionare l'informazione ritenuta rilevante e trarre valore da essa. Anche se le principali risorse web rilevanti sono recuperati, non sempre esse appaiono tra le prime visualizzate, e diventa difficile individuarle tra altre migliaia, in parte rilevanti.

**Richiamo scarso o assente:** Non sempre si ottengono risposte rilevanti per le nostre richieste, oppure pagine importanti e rilevanti possono non essere recuperate. Nonostante il basso richiamo sia il problema meno frequente con gli attuali motori di ricerca, esso potrebbe comunque verificarsi.

**Risultati molto sensibili al vocabolario:** Spesso la nostra parola “chiave” iniziale non ci fornisce i risultati sperati e non perché non ci siano documenti rilevanti, bensì perché questi ultimi, al loro interno, contengono una terminologia differente da quella utilizzata dalla query. Questo crea insoddisfazione perché query semanticamente simili dovrebbe fornire lo stesso risultato.

**I risultati sono singole pagine web**: Se noi abbiamo bisogno di un’informazione diffusa su numerosi documenti, siamo costretti a fornire ripetute query per collezionare documenti rilevanti ed estrarre manualmente la parziale informazione presente in ognuno per integrarla con il resto.

**Contenuti non sempre direttamente accessibili:** I risultati delle ricerche web, quando non sono semplici pagine, sono accessibili solo se si dispone di altri strumenti software (per esempio: .pdf, .doc, .xls, .ppt, .jpg ). Fortunatamente si assiste alla diffusione di prodotti open source generalmente gratuiti che consentono da parte dell'utente l'utilizzo concreto di tali contenuti.

Fare “information extraction” su una collezione così vasta e varia come quella dei documenti in rete è un compito particolarmente arduo. Il principale ostacolo per fornire supporto all'utente web, al momento, è rappresentato dal fatto che il significato del contenuto web non è accessibile alla macchina: naturalmente ci sono strumenti che possono recuperare testi, suddividerli in più parti, verificare lo spelling, contare i termini, etc, ma non sono ancora in grado di interpretare le frasi ed estrarre informazione utile per gli utenti, o almeno le capacità in tal senso sono per ora limitate.

Inoltre, per ottenere risultati soddisfacenti, è preferibile che l'informazione sia facilmente elaborabile dalla macchina: un testo strutturato in XML ad esempio.

Inoltre, per ottenere risultati soddisfacenti, è preferibile che l'informazione sia facilmente elaborabile dalla macchina: un testo strutturato in XML oppure opportunamente annotato è sicuramente più “comprensibile” per un elaboratore rispetto a un testo non strutturato.

1. **Il Web Semantico (Semantic Web Vision)**

*“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 should make 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, 1999)[[5]](#footnote-5)*

Il primo a usare l'espressione “web semantico” è stato Tim Berners: esso immagina computer capaci di analizzare tutti i dati presenti nel web, di interagire tra loro e di gestire molti aspetti della vita quotidiana. Che cosa si intende per “semantico”?

La semantica si definisce come “lo studio del significato delle parole33” Tuttavia, da una prospettiva ingegneristica, la semantica può essere intesa come “significato e uso di un'informazione”. E, infatti, la premessa di base del web semantico è far sì che ogni “frammento di dato” abbia un significato ben definito a scopo di utilizzo.

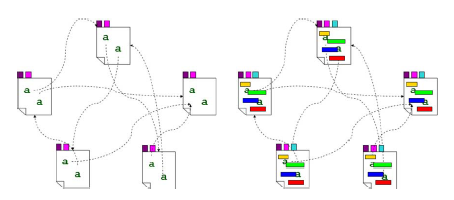
Con il miglioramento della tecnologia dei motori di ricerca, molte pagine in formato non HTML (PDF, DOC, XLS,...) sono state rese visibili ai motori di ricerca che possono individuarle e indicizzarle. Lo scopo del web “profondo”[[6]](#footnote-6) Consiste primariamente in questo: rendere i dati memorizzati nei database disponibili ed elaborabili da applicazioni web che li convertono dinamicamente (“al volo”) in pagine web.

Il Web Semantico può essere immaginato allora come un'estensione del web corrente in cui ai dati sono associate informazioni e significati ben definiti, per consentire alle persone e ai computer di lavorare meglio insieme.

*Oggi i contenuti web sono formattati per essere ben compresi dalle persone piuttosto che dai programmi e il linguaggio predominante con cui sono scritte le pagine web è HTML,il linguaggio che non prevede marcatori utili alla semantica. Un contenuto web semantico, invece, dovrebbe essere contemporaneamente comprensibile per l'utente e processabile per la macchina. Ciò richiede la possibilità di inserire varie annotazioni leggibili per l'utente e di specificare il significato del contenuto in modo preciso e disambiguo (un risultato si può ottenere tramite le varie specifiche XML come RDF). In pratica, il documento deve essere auto-descrittivo, e questa caratteristica si ottiene parzialmente producendo un linguaggio comune per specificare dati e metadati nel web. La standardizzazione è* *l'elemento chiave per migliorare la comprensione da parte degli elaboratori dei dati e contenuti del web ed è fattibile tramite l'uso di termini di base nella creazione di descrittori dei metadati.*

1. **Il modello dei dati nel web**

Se andiamo a considerare in modo molto semplificato il web come un insieme di documenti nei quali sono presenti alcuni metadati e hyperlink verso altri documenti (titolo, parole chiave, ecc), con collegamenti tra i documenti (vedere la parte sinistra della fig. 6). Che cosa fa il Web Semantico in questa immagine? Metadati semantici di vario genere, sia a livello di documento (i.e. hyperlinks) e sia a livello di contenuto (cioè attaccato a particolari parti e posizioni all'interno   
il documento). Come mostrato nella parte destra di *fig. 6*, dal punto di vista chiamato nel gergo a *volo d'uccello*, il Web semantico è più. colorato e ricco di quello attuale.



**Fig. 6:The current www (sinistra) e il Semantic Web (destra)**

1. **Dati, Metadati, RDF e simili**

Poiché la realizzazione di questa visione semantica del web dipende dalla presenza di una massa critica di metadati, l'acquisizione di questi metadati si pone come una sfida importante. Tenendo conto delle milioni di pagine web già esistenti , un accumulo di semantica esplicita non è considerata un approccio fattibile ,anche se in alcuni casi è inevitabile. La nostra visione è di fornire un sistema completamente automatico per l’annotazione semantica.

Con il nostro lavoro in questo progetto ha come obiettivo quello di creare, architettura scalabile per l'annotazione semantica automatica (JAPE) e attuare quest’architettura in una piattaforma basata su elementi per l'indicizzazione semantica per il recupero su grandi raccolte di documenti (GATE). Quello che è considerato il nostro contributo innovativo centrale è che offriamo un nuovo modello di tracciamento delle informazioni all’architettura già presente, per l’annotazione semantica automatica dei documenti.

L'informazione presente sul web può apparire sotto forma di dati *strutturati* (es. database relazionali), *semi-strutturati* (es. pagine scritte in XML), o *non-strutturati* (es. file di tipo .txt). Il concetto di dato è intuitivo: qualunque informazione rappresentata in modo da poter essere trattata da un calcolatore costituisce in pratica un dato.

Diverso invece il concetto di *metadato* che costituisce la caratteristica principale del web semantico e ha che il compito di descrivere il dato**: Semantic Web Content = METADATI + DATI.**

I *metadati* e le *annotazioni* si riferiscono al dominio specifico che vogliono descrivere e sono associati ai dati. Nel suo significato più generale, il metadato è definito come “informazione riguardante il dato”. Per database strutturati, il più comune esempio di metadato è lo schema del database. I metadati possono essere usati per memorizzare proprietà riguardanti il formato e sono utili nell'accesso o nel recupero dei dati. Essi possono descrivere il contenuto informativo oppure rappresentarne un sommario quando i dati sono descritti in maniera molto analitica. Sono utili, inoltre, per rappresentare proprietà o relazioni tra oggetti appartenenti a tipologie eterogenee.

**Si può utilizzare i soli metadati e hyperlink per realizzare una ricerca semantica? NO! Perché?**

Dal nostro punto di vista, al fine di individuare e definire il valore aggiunto dal Semantic Web, è fondamentale studiare di più la natura dei metadati semantici (Semantic Data). Supponiamo , di avere un tag < 2134 > in una parte del documento, come segue ". Abc < 2134 > xyz < / 2134 >. . . » . Questo metadato è utile ? Possiamo chiamarlo semantico ? Le risposte sono negative .

*Per avere metadati utili in un ambiente* *Web semantico*, *dovrebbe significare qualcosa*, *cioè i simboli (o referenze) che lo costituiscono dovrebbero consentire altre interpretazioni*. *I metadati dati cioè relativi “al dato”, sono dati che catturano parte del significato di un altro dato, da sola non è in grado di realizzare il “Web Semantic” ma è sicuramente un primo passo.*

*Attenzione però, è possibile OTTENERE UN DOCUMENTO STRUTTURATO CON I METADATI, il caso più semplice, ogni documento HTML può essere convertito nel suo relativo documento XML strutturato dai soli metadati, la soluzione è “robusta” ma costosa in termini di elaborazione.*

*Interpretazione è un approccio/elaborazione, che permette l'assegnazione di qualcosa di aggiuntivo ai simboli (cioè aggiunge valore alle informazioni), rispetto a un certo modello o ambiente. In una rappresentazione della conoscenza (KR) ambiente, questo è quello che è considerato un incarico di significato o semantica ai simboli, espressioni, ecc.*

*E ' importante rendersi conto che l'interpretazione è solo possibile rispetto a qualcosa, per qualche dominio, modello, ambiente (possibile) del mondo.*

Nei casi in cui il dominio d’interpretazione non è chiaramente definito, ciò significa che è ovvio e / o è troppo confuso e complesso per essere formalmente discusso. In letteratura, nel processo di scrittura, l’ambiente di interpretare ciò che viene scritto ed è totalmente aperto, gli autori scrivono spesso con l'intenzione di permettere molteplici (e anche ambigue) interpretazioni, questo consente una certa libertà di interpretazione al lettore. Nell'ambiente del Semantic Web , tuttavia , le interpretazioni dovrebbero essere eseguite automaticamente dalle macchine in stretta (e per quanto possibile) modo più prevedibile o intuitivo del lettore umano . Questo richiede una definizione formale d’interpretazione e, a causa di questo, una definizione formale dell’ambiente. Assumendo che uno stesso ambiente può essere modellati in modi diversi, allora esso permette differenti interpretazioni, che devono essere specificate in una concettualizzazione;

*Una concettualizzazione è un punto di vista astratto e semplificato del mondo che vogliamo rappresentare per qualche scopo .*

Tale concettualizzazione dove le *ontologie* (le descriveremo più avanti) cercano teorie logiche per la specificazione formale di una concettualizzazione , in questo modo , diventa evidente che non si può arrivare a utili applicazioni Web Semantico se si considera soltanto annotazioni in RDF ( S ) , OWL , o qualche altra lingua(fig.7) .

*Le seguenti risorse non saranno utilizzate ma è utile sapere qual e è il loro scopo:*

***RDF****: data model per oggetti (“risorse”) e relazioni tra loro. Fornisce una semantica semplice e i dati possono essere rappresentati in sintassi XML.*

***RDF Schema****: è un linguaggio per la descrizione del vocabolario, specifica proprietà e classi di risorse RDF, con una semantica basata su generalizzazione gerarchica.*

***OWL****: l'Ontology Web Language è un linguaggio per una descrizione “arricchita” del vocabolario, poiché specifica proprietà e classi come relazioni tra classi (p.e. “disgiunzione”), cardinalità (“esattamente uno”), uguaglianza, caratteristiche delle proprietà (p.e. “simmetria”) e classi numerate.*

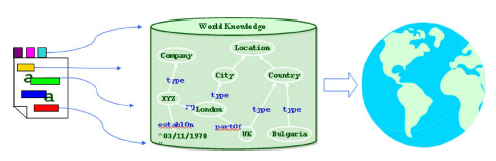
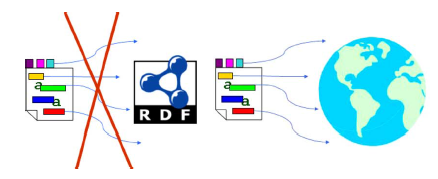
*Da sapere,* ***Resource Description Framework (RDF) e Web Ontology Language (OWL)*** *sono stati scelti dal progetto “Ontologie archivistiche” promosso dal Ministero per i Beni e le Attività culturali – Direzione Generale per gli Archivi – progetto che si propone come “sistema collaborativo di analisi e descrizione ontologica di sistemi archivistici nazionali e di una base concettuale condivisa"[[7]](#footnote-7)”, ideato e sviluppato in linea con le proposte del W3C per il Web Semantico". L'obiettivo fondamentale è la definizione di un'ontologia di due sistemi archivistici nazionali.[[8]](#footnote-8) E di un'ontologia “esterna” di concetti archivistici ritenuti rappresentativi della tradizione descrittiva nazionale, proposta e utilizzata come modello concettuale di riferimento rispetto al quale rapportarsi.*

*Le Annotazioni possono essere espresse in RDF (S), ma non è su RDF (S).* I metadati  
sono usati per fornire una formalizzazione parziale del contenuto dei documenti come prerequisito per una gestione più completa. Il contenuto del web è il prodotto della civiltà umana che si basa principalmente sul mondo e sul suo contenuto.

“*The content on the web, covers both phenomena in the real world or citizens in some more intangible mental spaces, indirectly referring to the reality.* “

(*A. Kiryakov*.)

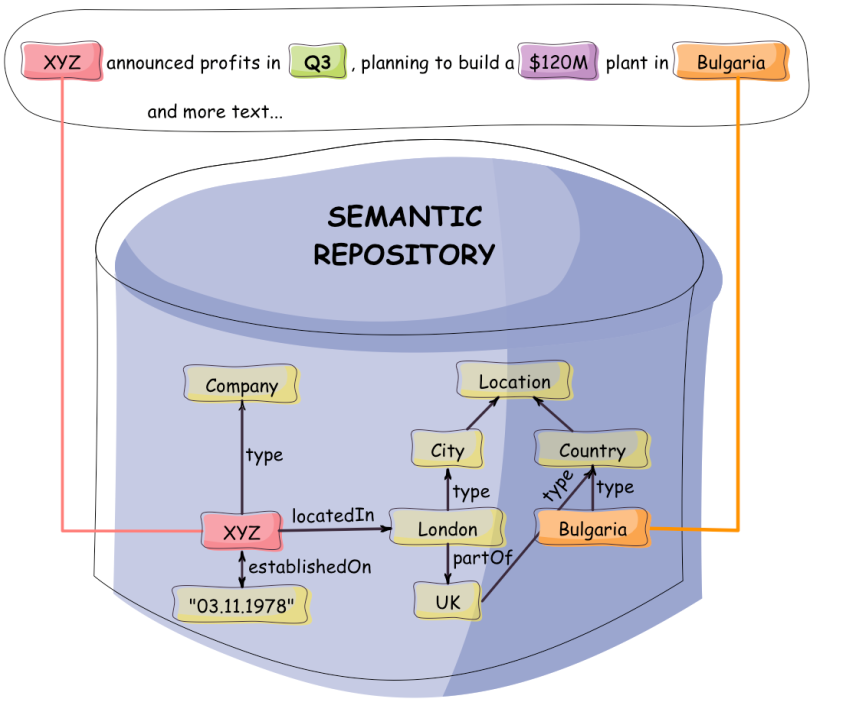
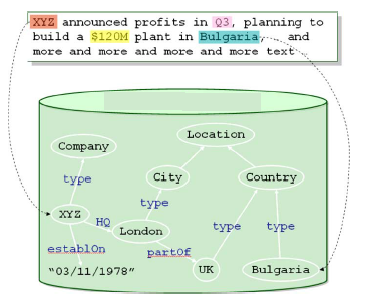
Sebbene quello che è stato detto di sopra è abbastanza informale e speculativo, lo riteniamo piuttosto importante, per la realizzazione di modelli nel Semantic Web, inoltre, come detto sopra, i metadati possono difficilmente fare riferimento (o essere interpretati) direttamente verso il mondo. Tali riferimenti non possono essere né formali né equivocabili. Esistono comunque strutture che prevedono l’utilizzo di metadati semantici per indicare direttamente è un modello formale del mondo (*fig. 7*) .



**Fig.7:Metada about the world not about the RDF (sinistra) ,Metadata referring to world knowledge (destra)**

1. **Le annotazioni semantiche**

L’Annotazione semantica è una generazione di metadati specifici e schema di utilizzo, al fine di attivare nuove informazioni dei metodi di accesso e di estendere quelle esistenti.  
In poche parole, annotazione semantica è l'assegnazione delle entità nei collegamenti di testo alla loro descrizione semantica ( come presentato in fig. 8 ) . Questo tipo di metadati fornisce sia informazioni di classe sia di istanza sulle entità . Se queste annotazioni sono chiamate " semantiche ", "entità" o in qualche altro modo, è tutta una questione di terminologia. Per quanto ne sappiamo, non esiste un termine ben consolidato per questo compito, né vi è un significato ben consolidato per il termine " annotazione semantica ". Ciò che è più importante è che le annotazioni semantiche automatiche consentono molti nuovi tipi di applicazioni: evidenziazione, indicizzazione e il recupero, la categorizzazione, la generazione di metadati più avanzati, “dribblando” tra testo non strutturato e conoscenza rilevante avviabile. L’annotazione semantica è applicabile per qualsiasi tipo di testo - pagine web, i documenti regolari (non web ), i campi di testo in banche dati ,ecc.



**Fig.8: Semantic Annotation**

1. **Le ontologie**

ll concetto più innovativo nel web semantico è quello di ontologia: il termine ha le sue origini nella filosofia e in tale contesto ci si riferisce alla natura dell'esistenza, “allo studio dell'essere in quanto essere”, come la definiva Aristotele o Sant'Anselmo d'Aosta.

Nell'ambiente della computer science, invece, ha assunto un significato diverso: anziché di “ontologia” si discute di “un'ontologia”. Alcune autorevoli definizioni recitano:

“L'ontologia è una specificazione esplicita e formale di una concettualizzazione”.[[9]](#footnote-9)

* Concettualizzazione: s’intende un modello astratto e semplificato del mondo che si desidera rappresentare per qualche scopo (dominio d’interesse). Ciò che esiste è esattamente ciò che può essere rappresentato.
* Esplicita: la tipologia dei concetti e le relative restrizioni devono essere ben definiti.
* Formale: deve essere machine-readable condivisa: cattura conoscenza consensuale non ristretta a qualche individuo.

“L'ontologia nella computer science è una rigorosa ed esaustiva organizzazione di un dominio della conoscenza che è generalmente gerarchica e contenente tutte le entità rilevanti di quel dominio e le loro relazioni”.[[10]](#footnote-10)

L’ontologia può essere dunque pensata come un insieme di termini e relazioni che denotano i concetti rappresentativi di un particolare dominio d’informazione; essi possono caratterizzare la conoscenza in un’applicazione, in un dominio specifico o generalizzato (upper ontologies). L'ontologia descrive una particolare conoscenza in modo univoco: non esiste ambiguità poiché termini e relazioni sono condivise dall’intera comunità di utenti del dominio di riferimento e, al tempo stesso, assegna ai termini significati “machine-readable”.

Le definizioni sopra citate indicano contestualmente le principali fasi per costruire un’ontologia: integrandole opportunamente, si possono così riassumere:

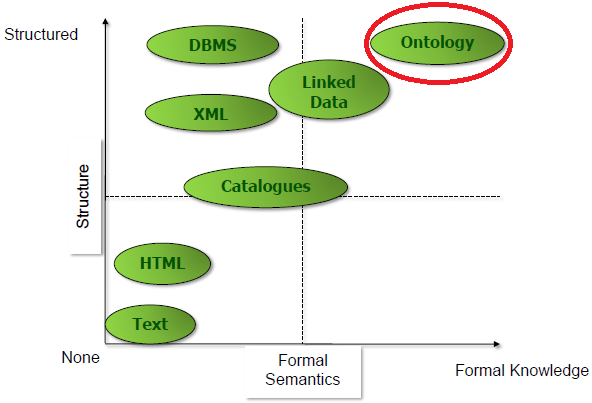
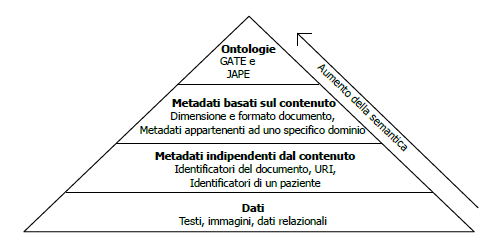
* Determinazione di un ambito (dominio);
* Valutazione di un eventuale riutilizzo, anche parziale, di ontologie già esistenti;
* Identificazione dei termini base;
* Definizione della tassonomia (gerarchia dei termini e loro relazioni);
* Definizione delle proprietà e delle regole (specificandone dominio e range);
* Individuazione di nuovi termini prodotti dalle regole;
* Definizione d’istanze (popolamento dell'ontologia);
* Identificazione di eventuali inconsistenze.

Dunque, un’ontologia non include solo termini esplicitamente definiti in essa, ma anche quelli che possono essere derivati usando tali regole.

Un’ontologia può presentarsi in varie forme ma include sempre un vocabolario di termini e una descrizione dettagliata del loro significato. La sua caratteristica principale è la concettualizzazione di un dominio, ossia una rappresentazione formale della realtà così com’è percepita e comunemente condivisa. Questo significa che, con riferimento ad una specifica realtà si possono avere vocabolari diversi, ma un'unica concettualizzazione di quella realtà.

Ontologie e istanze formano la *knowledge base*: i termini definiscono concetti importanti (classi di oggetti) del dominio. Le relazioni generalmente includono gerarchie di classi (X *subClassOf* Y), ma anche proprietà (X *cause* Y), vincoli sul valore (X hasValue Y), statement disgiunti (X is\_a Y), specifiche sulle relazioni logiche tra oggetti. In pratica, nell’ambiente web, le ontologie forniscono una comprensione condivisa di un dominio, superando le differenze nella terminologia.

Riassumendo quanto finora visto, possiamo quindi immaginare una stratificazione come quella illustrata di seguito (fig.9), la base è costituita dai dati, ossia da tutto ciò che è oggetto di elaborazione; al livello superiore si possono collocare i metadati indipendenti dal contenuto che aggiungono informazione ai dati, ma con un apporto semantico ancora molto ridotto; i metadati basati sul contenuto rappresentano il primo livello di “interpretazione” dei dati, tentando di descrivere in maniera strutturata il contenuto di un documento o di fornire informazione con riferimento ad una precisa realtà; al vertice sono presenti le ontologie, punto di arrivo per la concettualizzazione di un dominio specifico ma anche punto di partenza per ontologie superiori e più complesse, indipendenti dal dominio.



**Fig.9: Immagine rielaborata da V.Kashyap, C.Bussler, M. Moran - “The Semantic Web” - Springer 2008**

In quest’ambiente, la standardizzazione dei contenuti dell’informazione è cruciale: infatti, la mancanza di un’interpretazione condivisa porta a una limitatezza di comunicazione sia tra le persone, sia tra gli applicativi software, a scapito dell'interoperabilità, della condivisione e del riutilizzo.

Le ontologie sono molto utili per l'organizzazione e la navigazione dei siti web, essi espongono sulla sinistra della pagina i top level di una gerarchia concettuale di termini; cliccando su di essi, l'utente espande le sottocategorie e accede alle varie risorse disponibili.

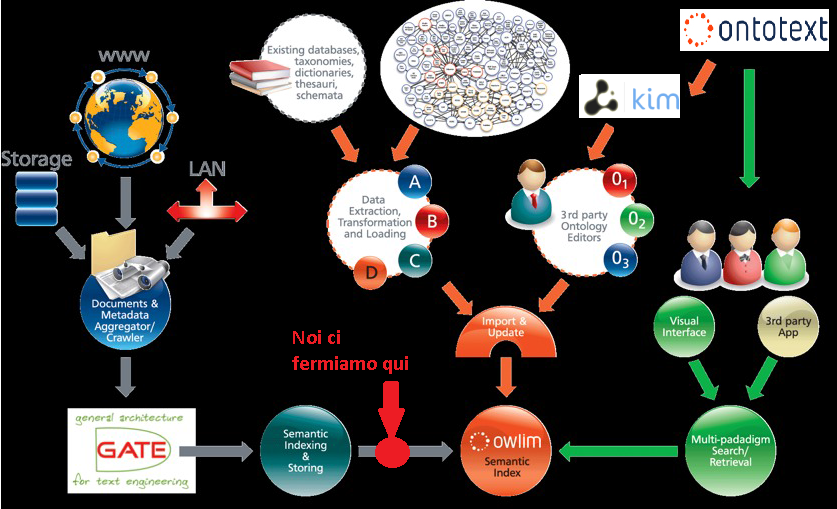
Le ontologie sono molto utili anche per migliorare l'accuratezza delle ricerche web, i motori di ricerca possono cercare pagine che si riferiscono a un preciso concetto presente in un’ontologia invece che collezionare tutte le pagine nelle quali certe keywords sono presenti. In questo modo le differenze nella terminologia tra pagine web e query possono essere superate.

I motori di ricerca, in più, possono sfruttare la generalizzazione o la specializzazione dell'informazione.

Se una query non restituisce alcun documento rilevante, il motore di ricerca potrebbe suggerire all'utente una query più generale. Oppure se i risultati sono troppi, il motore potrebbe suggerire una ricerca più raffinata. La descrizione delle Ontologie nel Web è affidata a linguaggi specifici RDF, RDF Schema, OWL.

1. **Conclusione sulla semantica Web**

Ci sarebbero molti altri concetti fondamentali e problemi di cui parlare, ma nel nostro progetto ci interesseremo solo alla creazione di un modello per delle annotazioni semantiche, all’estrazione del loro contenuto e allo “storage” delle stesse su un database opportunamente strutturato, senza entrare troppo nel dettaglio dell’argomento. Per realizzare tale modello utilizzeremo la **Semantic Annotation API per eccellenza GATE**. Nel grande viaggio del web semantico noi ci fermeremo a una delle prime fermate.



**Fig.10:Rappresentazione più o meno reale , del Web Semantico attuale(2013-2014).**

1. **GATE -General Architecture Text Engineering[[11]](#footnote-11)**

**GATE** può essere pensato come un'architettura sofware per il Language Engineering (LE). A sua volta il LE può essere definito come “la disciplina o l'arte dei sistemi software in grado di eseguire task che implica l'elaborazione del linguaggio umano, sia i processi di costruzione sia i suoi output sono misurabili e predicibili.”[[12]](#footnote-12)

L’ambiente in cui s’inserisce è più chiaro se si considerano le seguenti definizioni:

* **Computational Linguistics (CL)**: scienza del linguaggio che usa il calcolo come strumento investigativo.
* **Natural Language Processing (NLP)**: scienza che si occupano di strutture dati e algoritmi per elaborare il linguaggio umano.
* **Language Engineering (LE)**: costruisce sistemi NLP i cui costi e risultati siano misurabili e predicibili.

GATE si pone come Software **Architecture for Language Engineering (SALE)**, ossia come infrastruttura software per integrare e sviluppare strumenti per CL, NLP e LE.

Come definito nel sito stesso, cui si rimanda per ogni approfondimento, GATE è un software open source che comprende un ambiente di sviluppo integrato (IDE - Integrated Development Environment) per l'utilizzo di componenti di NLP già presenti nell'applicativo e un insieme di utili plugin da installare all'occorrenza. Le principali funzionalità di GATE comprendono tokeniser, gazetteer, sentence splitter e parser; l'applicativo fornisce inoltre strumenti per analisi di testi scritti in varie lingue sebbene, purtroppo, non abbia ancora un plugin per i testi in italiano, consente di utilizzare algoritmi di machine learning e di costruire ontologie. In particolare fornisce un sistema di IE, chiamato ANNIE, che comprende una serie di elementi normalmente utilizzati in una tipica elaborazione di NL e che si sono rivelati particolarmente utili. Per il test è stata utilizzata la versione di GATE 7.1, scaricata dal sito ufficiale.

Prima di descrivere i moduli utilizzati e il procedimento attuato, si definiscono i seguenti concetti su cui si basa GATE, necessari per la comprensione di quanto descritto in seguito:

* **Applications Resources (AR)**: consentono di definire sequenze di processing resources che possono elaborare singoli documenti oppure l'intero corpus.
* **Language Resources (LR)**: consentono di visualizzare il documento o il corpus sul quale si sta lavorando.
* **Processing Resources (PR)**: visualizzano i moduli attivati dall'utente che possono essere selezionati e inseriti nelle sequenze di cui al punto uno.
* **Datastores (DS)**: rende persistente il corpus e le annotazioni create dall'utente sui documenti mediante memorizzazione in un datastore.
* **Document** (Documento): elemento testuale oggetto di elaborazione.
* **Corpus o corpora**: set di documenti che saranno elaborati in modo uniforme dai vari moduli selezionati dall'utente.
* **Annotation**: annotazione che sarà creata analizzando il documento.
* **Annotation set**: gruppo di annotazioni tra loro correlate**.**

Forniremo una breve introduzione italiana degli elementi base del software per maggiori dettagli consultare la documentazione ufficiale[[13]](#footnote-13).

1. **Generazione dei Corpora**

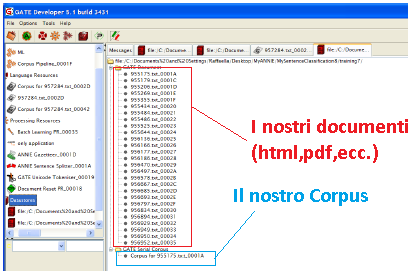
La schermata di avvio si presenta così:



**Fig.11:Ambiente di lavoro di GATE**

Come si può notare, la grafica è molto essenziale: le poche icone presenti sulla destra e in alto consentono di attivare i singoli elementi di GATE (AR, LR, PR, DS) e di aggiungere eventuali plugin.

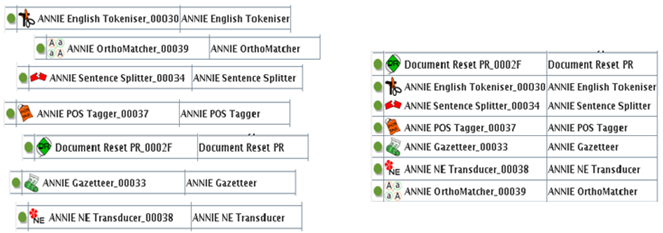
Innanzitutto è necessario caricare i documenti da elaborare: selezionando LR, si carica un documento (nel nostro caso un referto), si crea un corpus contenente quel referto e infine si popola il corpus aggiungendo tutti gli altri referti. Si ottiene così un insieme di documenti gestiti come un unico grande documento. Per far sì che le annotazioni create sui documenti rimangano “persistenti”, ossia memorizzate, è necessario definire un **datastore**: delle due tipologie possibili, è stato scelto un SerialDataStore basato su un sistema di serializzazione di Java. All'utente, il datastore appare come due directory destinate a contenere copia dei documenti annotati e l'elenco degli stessi denominato **Corpus** .



**Fig.12:Creazione del Corpus di Documenti in GATE**

1. **ANNIE – A Nearly New Information Extraction System**

GATE include un sistema completo di Information Extraction chiamato ANNIE (a Nearly-New Information Extraction System). Tale sistema non è caricato automaticamente dal programma ma deve essere selezionato dall'utente come plugin, configurato ed eventualmente stabilito per il caricamento automatico a ogni avvio di GATE. Esso comprende una serie di moduli di seguito brevemente illustrati perché utilizzati per il test.

****

**Fig.13: Elementi di ANNIE forniti di default (sinistra),Pila ordinata di default di ANNIE(destra).**

**NOTA: La regola JAPE nella pipeline è il nostro file JAPE MULTIFASE.**



1. **Document Reset[[14]](#footnote-14)**

Questa risorsa consente di resettare le annotazioni aggiunte al documento nelle varie elaborazioni ed è particolarmente utile quando si eseguono ripetute elaborazioni sullo stesso corpus per sperimentare i differenti risultati; restano invariate eventuali annotazioni originali. L'utente ha facoltà di impostare dei parametri per decidere un reset parziale delle annotazioni.

Tale risorsa è stata inserita in ogni test prima di utilizzare le altre risorse PR in modo tale che le annotazioni aggiunte da elaborazioni precedenti non influenzino i risultati ottenuti man mano.

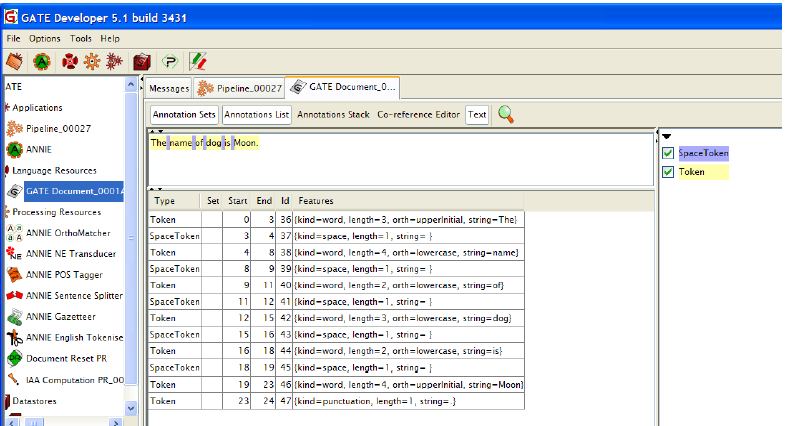
*GATE mantiene annotazioni e testo separatamente (StandOffMarkup), di fatto qualsiasi documento GATE passerà attraverso un’inalterata pipeline, in altre parole nessuna annotazione o parsing d’informazioni è incorporato nel documento. Per ragioni di design che non ci interessano, GATE tende a usare un riferimento debole alla JVM, producendo un overriding del Garbage Collection di conseguenza avremo bisogno di risorse che esplicitano direttamente i documenti per lo più quando si utilizza GATE programmaticamente. GATE terrà inoltre set di annotazioni derivate dalle elaborazioni precedenti nella memoria. Questo a due conseguenze:*

* + - 1. *Vecchie annotazioni residenti in memoria da un precedente documento possono essere prese in considerazione per l'elaborazione del documento successivo.*
      2. *L'heap JVM comincerà a riempirsi di vecchi set di annotazione generati   
         da precedenti documenti che possono incidere pesantemente sulle prestazioni. Lo scopo del Documento di reset PR è quello di pulire ogni precedenti annotazioni, essenzialmente rilasciando gli oggetti set di annotazione per il garbage e altro…..*

1. **Tokenizer[[15]](#footnote-15)**

Scopo di questa risorsa è suddividere il testo in “pezzi” come numeri, punteggiatura e parole. Come output si ottengono annotazioni di tipo SpaceToken oppure Token cui sono associati di default gli attributi come string, orth, category, length, etc.

Per esempio, se il documento da processare è “The name of dog is Moon.”, il tokeniser restituirà i valori evidenziati nella seguente schermata:



**Fig.14:Risultato del Tokenizer.**

Gli attributi associati ai Token e agli SpaceToken hanno i seguenti significati:

* ***kind****:* per distinguere se si tratta di *word, number, space* o *punctuation;*
* ***length****:* per indicare la lunghezza in caratteri del token
* ***orth****:* per indicare com’è scritto il *Token.* Può avere i seguenti valori:
  + ***lowercase***: se tutte le lettere sono scritte in minuscolo
  + ***upperInitial***: se la prima lettera è maiuscola e le successive minuscole
  + ***allCaps***: se tutte le lettere sono maiuscole
  + ***mixedCaps***: tutte le altre combinazioni non previste dalle prime tre
* ***string****:* stringa che riporta i caratteri di cui si compone il *Token.*

**GATE offre due tipi di tokeniser: quello fornito di default da ANNIE, ANNIE English Tokenizer e uno più generico (GATE Unicode Tokeniser)**. Il primo è utilizzabile esclusivamente per testi in lingua inglese perché riconosce particolari regole grammaticali, tra cui le forme contratte. Per tutte le altre lingue, è necessario usare quello generico il **GATE Unicode Tokenizer** che è valido per tutte le lingue occidentali.

1. **Sentence Splitter[[16]](#footnote-16) e RegEx Sentence Splitter[[17]](#footnote-17)**

La PR Annie **Sentence Splitter** divide il documento elaborato in segmenti corrispondenti a frasi. Essa aggiunge due annotazioni di facile comprensione: ***Sentence***(che individua il testo della frase) e ***Split***(che rappresenta il punto di fine frase)*.*

Esiste anche un modulo alternativo: ***RegEx Sentence Splitter*** che basa il suo funzionamento sulle espressioni regolari cioè su una struttura ben definita delle frasi in molti casi quando dobbiamo integrare una lingua non supportata pienamente da GATE è consigliabile utilizzare questa seconda scelta.

Attenzione: Dato un *RegEx Sentence Splitter,*se si prova a testarlo più volte su un documento, può rendere il sistema instabile vuol dire che dovete migliorare l’espressione regolare e i vari file di lexicon cui fa riferimento o utilizzare il Sentence Splitter di ANNIE che è “accettabile” per tutte le lingue occidentali.

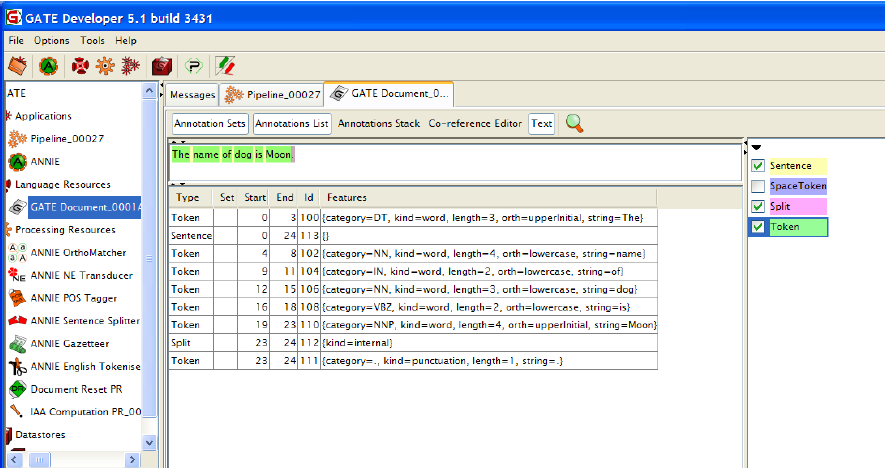


1. **Part of Speech Tagger [[18]](#footnote-18) e Generic Tagger[[19]](#footnote-19)**

L'effetto di questa PR sul testo considerato, consiste (utilizzando un lessico di default e un set di regole), nell'assegnazione a ciascun Token di un altro attributo ***Category*** (i tag di POS) cioè l'assegnazione di una categoria linguistica, valorizzato in conformità a una codifica predefinita la cui lista completa[[20]](#footnote-20) delle categorie è disponibile sull’user guide di GATE e di cui si riportano a titolo di alcuni esempi:

* **DT determiner** – Articles including ‘a’, ‘an’, ‘every’, ‘no’, ‘the’, ‘another’, ‘any’, ‘some’, ‘those’.
* **NN noun** – singular or mass
* **IN** – preposition or subordinating conjunction
* **VBZ verb** – 3rd person singular present
* **NNP proper noun** – singular (all words in names usually are capitalized but titles might not be), literal period
* **NNPS proper noun no singular** – plural

*Sono presenti anche molte regole JAPE di terze parti che realizzano categorie personalizzate compatibili con GATE e che possono essere utilizzate da chiunque, per il nostro progetto ne faranno poco utilizzo.*



**Fig.15:Risultati ottenuti applicando Tokenizer+Sentence Splitter + POS Tagger**

Come si può facilmente intuire questa funzionalità è molto utile se si utilizzano documenti scritti in lingua inglese mentre non hanno senso applicarla su testi in lingua italiana in quando darebbe risultati inaffidabili. Inoltre, come già accennato, per l'italiano esistono al momento solo un servizio web di POS tagging e non un modulo integrabile con GATE. Tuttavia, dato che sono già disponibili dei plugin per alcune lingue europee diverse dall'inglese, si spera che lo staff di GATE rilasci a breve anche quello specifico per i testi in italiano.

*Esistono numerose versioni di questa funzionalità, tuttavia sono sempre sviluppate per la lingua inglese e, in rari casi, ad altre lingue. Per la lingua italiana per chi si volesse “divertire” c’è, il Tanl Italian Parser[[21]](#footnote-21), dell’Università degli Studi di Pisa per ora fornito solo come servizio web ma che è molto potente ,noi ci limiteremo a usare un file .par. Inserendo una frase, si ottiene l'immediato parse tree.*



**Fig.16:Anteprima del potente Tanl Italian Parser dell’Università di Pisa**

*Anche in questa fase l'ambiguità, stavolta di tipo sintattico (detta anche ambiguità locale perché riferita a una parte della frase), comporta non poche difficoltà. Si consideri questo esempio:*

*“Egli vide una donna con gli occhiali”*

*Non è ben chiaro se l'uomo ha gli occhiali e vede una donna oppure se la donna vista dall'uomo indossa gli occhiali. A volte l’ambiente aiuta a risolvere tali ambiguità, ma alla presenza di frasi articolate si genera una proliferazione di alberi sintattici con evidente aumento della complessità.*



1. **Gazetteer[[22]](#footnote-22)**

Il ruolo di questa PR è identificare entità nel testo basandosi su apposite liste. Questa fase, da un punto di vista pratico, non presenta particolari difficoltà; tuttavia richiede notevole accuratezza nella predisposizione delle liste, se si vogliono ottenere risultati soddisfacenti.

Fornito un documento o referto contenente per puro caso i nomi di sette regioni italiane, se si volesse raccogliere tutte le informazioni sulle regioni, in questo caso il nome, una possibile soluzione potrebbe essere quella di utilizzare corrette liste di Gazetteer con estensione .lst (le vedremo in maggior dettaglio nella *Customizzazione del Gazetteer*), ottenendo già un primo livello di struttura e delle prime annotazioni che vanno a evidenziare semplicemente gli elementi delle liste (.lst).

Il vantaggio, già a questo livello, è quello di ottenere in modo agevole e “abbastanza” veloce una serie di dati utili per impostare i primi criteri di analisi. Per esempio, è possibile selezionare solo i documenti che, nel loro testo, fanno riferimento alla regione\i. Tale approccio, tuttavia, richiede un'analisi non banale relativa alla scelta più opportuna delle annotazioni e degli attributi a esse associati, soprattutto se si pensa a una loro combinazione per dedurre altri elementi informativi.

Per quanto concerne la classificazione che si vuole ottenere, non è richiesto un livello di annotazione dettagliato come nell'POS Tagger ma sono comunque necessarie alcune accortezze. Di sicuro il termine “Toscana” può essere individuato anche dalla sequenza di parole “la regione Toscana”: dunque è più corretto individuare quest'ultima sequenza, quando presente, che non il singolo termine. Se s’immagina poi di avere a disposizione tutti i nomi delle regioni, delle provincie e delle città italiane, indipendentemente dalla terminologia usata, allora diventa fondamentale fornire al sistema un elenco completo di termini “equivalenti”. In questo caso forse con i nomi delle regioni è difficile visualizzare bene qual è l’equivalente di Toscana, immaginate il nome di una malattia la “Varicella “ in molti testi, di medicina è indicata con il nome “Varicellovirus” è chiaro che sono due termini che indicano la stessa cosa quindi “equivalenti” e che molto probabilmente rientreranno nella stessa lista Gazetteer delle malattie.

Tutti i file creati dall'utente (ed eventuali resi disponibili da GATE che però in questa circostanza non sono utilizzabili visto che i referti sono scritti in italiano e non in inglese) devono essere specificati in un file riassuntivo, accoppiando a ciascuno nome una variabile, attraverso un file di definizione che è indicato con l’estensione .def. Le variabili saranno utilizzate dalla successiva PR. L'esecuzione di ANNIE Gazetteer comporta la creazione di annotazioni di tipo **Lookup** con un attributo di tipo **majorType** che prenderà il valore della variabile da noi specificata. Maggiori dettagli a Customizzazione del Gazetteer.

1. ** Jape Transducer[[23]](#footnote-23) o ANNIE NE Transducer[[24]](#footnote-24)**

**ANNIE NE Transducer e Jape Transducer** entrambi svolgono lo stesso lavoro sceglietene uno.

Si consideri ora la fase che utilizza i Lookup ottenuti con ANNIE Gazetteer. JAPE (Java Annotation Patterns Engine) è un linguaggio pattern-matching, una modalità che consente di scrivere alcune regole in base ad una precisa sintassi, che poi saranno tradotte in codice JAVA ed eseguite secondo una precisa sequenza, producendo come output ulteriori annotazioni, modifiche su quelle esistenti oppure modifica/creazione di attributi.

Una JAPE grammar consiste in un main.jape in cui si specifica secondo un preciso ordine logico un set di fasi. La sequenza di fasi è giustificata dalla necessità di creare annotazioni temporanee che si trasformano in input nelle fasi successive per essere combinate tra loro o trasformate. Ciascuna fase è costituita da regole che individuano dei pattern ed eseguono delle azioni su di essi e poiché sono proprio le regole a dare significato ai dati finora estratti, se ne descrive il funzionamento a grandi linee.

* Tutto quello che fa JAPE, può essere svolto con JAVA
* Si può creare pattern JAPE in LHS complicati e blocchi JAVA in RHS semplici, oppure il contrario, dipende da quello che vogliamo.
* JAPE è stato creato e studiato per riuscire ad assegnare valore semantico e coreferenze fra le parole in modo semplice e intuitivo ma non ha un debug ottimo.

**Per comprendere appieno le regole JAPE si fornisce una piccola introduzione italiana alla programmazione delle regole JAPE all’APPENDICE di questo documento.** Mostriamo un esempio di regole JAPE in cui creiamo un semplice pattern per la cattura degli elementi della lista regioni italiane in un documento di testo:

**/\*\*Fase della regole JAPE essenziale per essere richiamata dalla nostra main.jape\*/**

**Phase: regioni**

**/\*\*Fornisco come Input il tag Lookup e la variabile VarRegioni\*/**

**Input: Lookup VarRegioni**

**/\*\*Opzione di controllo (facoltativa)\*/**

**Options: control = appelt**

**/\*\*Il nome della regola\*/**

**Rule: Regione**

**(**

**/\*\* In pratica gli sto dicendo "evidenzia sul testo del documento tutti gli elementi presenti nella lista" identificata dala**

**\* variabile VarRegioni (maggiori detttagli nella customizzazione del gazetteer più avanti) \*/**

**{Lookup.majorType==VarRegioni}**

**):name\_regione**

**-->**

**:name\_regione**

**{**

**/\*\*Questo pezzo di codice JAVA utilizza le librerie di GATE, per comprenderlo consultare la documentazione ufficiale (maggiori**

**\*dettagli nella customizzazione delle regole JAPE più avanti)\*/**

**gate.AnnotationSet matchedAnns= (gate.AnnotationSet) bindings.get("name\_regione");**

**gate.FeatureMap newFeatures= Factory.newFeatureMap();**

**newFeatures.put( "rule" , "Regione" );**

**annotations.add( matchedAnns.firstNode(),matchedAnns.lastNode() , "Regione", newFeatures );**

**}**

**Fig.17:Un’esempio di regola JAPE per il tracciamento dei nomi delle regioni italiane**

Lo input Lookup abilita il funzionamento del Lookup nel documento, lo input VarRegioni identifica l’utilizzo delle liste (.lst) delle regione italiane secondo la variabile che gli è stata assegnata dal file riassuntivo (.def) in questo caso VarRegioni(maggiori dettagli a Customizzazione del Gazetteer).

Nel caso considerato, cercherà tutti i Lookup il cui attributo sia uguale a VarRegioni, creando la variabile name\_regione.

Tutte le corrispondenze trovate sulla base delle specifiche fornite dall'utente nella LHS sono “passate” alla RHS tramite la variabile name\_regione e “depositate” in un oggetto Java di tipo AnnotationSet. Da questo set, si estraggono man mano i dati contenuti e a ognuno si associa una nuova annotazione completa di eventuali attributi a essa collegati (nel nostro caso crea l'annotazione Regione con attributo rule). A volte è utile impostare come valore dell'attributo la stringa stessa cui si riferisce l'annotazione. In questo caso la RHS estrae i valori dell'offset iniziale e finale che identificano la posizione della stringa all'interno del documento, e copia nell'attributo i caratteri della stringa. L'esempio precedente si trasforma quindi come segue:

**:name\_regione**

**{**

**gate.AnnotationSet matchedAnns= (gate.AnnotationSet) bindings.get( "name\_regione" );**

**//estrai i pattern richiesti dalla LHS della regola**

**int begOffset = matchedAnns.firstNode().getOffset().intValue();**

**int endOffset = matchedAnns.lastNode().getOffset().intValue();**

**//ricava il valore degli offset in base alla posizione della stringa nel testo**

**String myDoc = doc.getContent().toString();**

**String matchedString = myDoc.substring(begOffset, endOffset);**

**//usa gli offset ottenuti in precedenza per estrarre la stringa di testo**

**gate.FeatureMap newFeatures= Factory.newFeatureMap();**

**//crea un nuovo attributo “valore” = “stringa”**

**newFeatures.put("kind" , matchedString);**

**//crea un nuovo attributo “rule = “nome della rule"**

**newFeatures.put("rule" , Regione) ;**

**//aggiungi un'annotazione “disease” con attributo “valore” uguale a “stringa”**

**annotations.add(matchedAnns.firstNode(),matchedAnns.lastNode(),"Regione" , newFeatures);**

**}**

**Fig.18:Riprendendo la regola JAPE precedente si va a immettere le informazioni annotate in un’apposito feature**

*Perché andiamo a mettere i valori delle nostre annotazioni in una FeatureMap? Non potremmo richiamare direttamente il valore della stringa che le nostre annotazioni hanno matchato? Certo che si! Lo si può fare anche attraverso le Feature, l’API di GATE che vedremo meglio in dettaglio più avanti fornisce pro e contro se utilizzare o meno una FeatureMap per ottenere le informazioni annotate del documento o utilizzando direttamente le annotazioni. Nel paragrafo della customizzazione delle regole JAPE vedremo un esempio completo e dettagliato.*

*Se si specifica in ingresso delle annotazioni non di default, tutte le annotazioni di default cioè Lookup, Token, ecc. saranno ignorate , esse sono annotazioni che avremmo solo generalizzato ulteriormente per delle regole che non verrebbero mai lanciate.*

*E’ possibile definire più regole JAPE nel medesimo file.jape, quindi più regole JAPE che lavorano nella medesima fase (Phase) e che devono essere coordinate con gli attributi di priorità (Priority), controllo (options), perché farlo ? perché complicarci la vita con il rischio di avere una collisone delle fasi tra le regole? Se abbiamo una regola JAPE già definita che “cattura” i nomi propri delle persone nel documento creando la sua personale annotazione temp. In una regola successiva (sia che lavori nella medesima fase basta che sia chiamata dopo o in una fase successiva), con cui vogliamo catturare il nome e cognome delle persone, riutilizziamo la regola già definita per i nomi con un comando di Lookup all’annotazione precedente.*

1.  **Ortomatcher[[25]](#footnote-25)**

Il modulo Orthomatcher aggiunge rapporti d’identità tra entità nominate trovati dai Semantic Tagger (le nostre regole Jape), al fine di eseguire coreference (co-referenze). Esso non trova nuove entità denominate perché tali, ma può assegnare un tipo, un nome proprio non classificato (un'annotazione sconosciuta), utilizzando il tipo di un nome corrispondente.   
  
Le regole di corrispondenza sono richiamate solo se i nomi confrontati sono entrambi dello stesso tipo, o se uno di essi è classificato come 'sconosciuto'. Questo impedisce a un nome in precedenza classificato che venga ri-categorizzato.

**i.e** siano già etichettati come le ‘Organization’, o se uno dei essi è classificato come 'sconosciuto'. Questo impedisce a un nome precedentemente classificato di essere categorizzato di nuovo.

Un elenco di alias esiste anche per Orthomatcher:

**i.e** Il PR crea un rapporto di identità tra il "Big Blue" e "IBM". L'alias Lista tuttavia può essere costruito manualmente come accennato in precedenza.

1. **Considerazioni sull’uso di GATE**

La scelta di GATE è già stata motivata: una presentazione dell'applicativo ben fatta, un sito internet ben strutturato nella forma e ricco nei contenuti, un'impressione generale positiva e, soprattutto, una documentazione corposa e facilmente consultabile, hanno dirottato la scelta verso questo open source. Purtroppo il suo utilizzo è stato fin dall'inizio difficoltoso: la grafica è a volte “troppo” semplice nel senso che non permette un uso intuitivo dell'applicativo ma richiede una lettura attenta e rigorosa della documentazione.

Altra nota dolente è proprio la documentazione: in alcuni punti è chiara e ben comprensibile, in altri fornisce informazioni essenziali, ma senza esempi a supporto o appena accennati. Capita allora la necessità di dedicare molto tempo, a tentativi frustranti per cercare di far funzionare una risorsa secondo le esigenze dell'utente. Per esempio, alcune istruzioni JAPE indicate nella documentazione e descritte con un esempio, funzionano bene con una singola condizione, ma se l'utente ha necessità di specificarne più regole, da loro stessi definite, rischia di dedicare ore a innumerevoli tentativi pensando si tratti di un errore di sintassi e invece di un preciso vincolo non dichiarato. Quando poi scopre che è un problema noto e molti altri hanno avuto la stessa difficoltà, si auspica che la documentazione possa essere integrata quanto prima. I moduli di ANNIE (tokeniser, gazetteer, sentence splitter....) sono ben documentati ma solo r incorrendo ad un tutorial non incluso nella guida si è potuto ordinarli nella corretta sequenza. Molto scomoda si è rivelata la scrittura delle regole in JAPE: trattandosi di testo scritto con Notepad++ o simili, non prevede un editor specifico né un debug che aiutasse nell'individuazione degli errori.

Una volta acquisita pratica, l'applicativo si rivela un buono strumento di NLP e un suo punto forte è sicuramente la ben dotata libreria di plugin. Sarebbe interessante provare in futuro prodotti alternativi per confrontarne la facilità d'uso e il modo di funzionare offerte. L'esperienza comunque ha dato la possibilità di approfondire argomenti mai affrontati in corsi universitari e, da un punto di vista personale, molto interessanti.

Per mancanza di tempo, in parte dovuta proprio alle difficoltà incontrate nell'uso dell'applicativo, non si è potuto approfondire la funzionalità relativa alle ontologie: GATE offre la possibilità di crearle e uno sviluppo futuro potrebbe essere proprio un'elaborazione dei documenti tramite le ontologie.

1. **Le annotazioni di GATE e HTML**

I file HMTL costituiscono la maggioranza dei documenti presenti nel web, sono caratterizzati da una taggatura particolare per lo più prevista da GATE di default, se si va a far analizzare a GATE un file HTML, esso fornisce già le annotazioni per il tracciamento delle varie parti del documento div, head, tutina, footer, ecc. (Original Markup) di default. Questo lo rende uno strumento molto potente perché può semplificare di molto la ricerca delle informazioni evitandoci di dover elaborare documenti anche molto grossi. Le annotazioni di GATE utilizzano questo sistema per estrarre ad esempio informazioni presenti nel solo <HEAD> o nel <FOOTER> del documento. Vedremo di seguito diversi metodi per il tracciamento delle varie zone di un documento HTML.

Un documento non HTML (doc, pdf, ecc.) è elaborato da GATE come un documento HTML costituito dal solo BODY, notando che assegna comunque il tag HEAD, FOOTER ma senza alcun contenuto.

Vedremo tre metodologie per relazionare le annotazioni con gli elementi dei documenti HTML:

1. Con le regole Jape.
2. Con la componente Annotation Set Transfer.
3. Con Java.
4. **Con le regole Jape**

Una volta capito come muoversi, sono davvero semplici, ma a volte possono entrare in contrasto tra di loro per cui occorre fare attenzione nel loro utilizzo mostriamo un esempio per tracciare il BODY e l’HEAD.

**ATTENZIONE**: per funzionare gli input Token e div,footer,ecc. devono essere nello stesso AnnotationSet questo possibile farlo con l’Annotation Set Transnfer che descriviamo al punto 2.

**Phase: html\_footer**

**Input:Token Lookup Split SpaceToken div footer head body h1 h2 federico /\*Ricordate di mettere gli input\*/**

**Options: control = all**

**Rule: nome\_regola**

**Priority:10**

**(**

**(Token within {footer}) /\*prende tutti i token all’intenro del tag footer del documento HTML\*/**

**(Token within {head}) /\*prende tutti i token all’intenro del tag head del documento HTML\*/**

**({Token within {div.id== ”federico” }}) /\*prende tutti i token all’interno del tag div con id federico del documento HTML\*/**

**):temp -->**

**{blocco\_java}**

**Fig.19: Regola Jape dimostrativa su come lavorare sui tag HTML di default di GATE**

****

**Fig.20:Il risultato della regola Jape su un semplice documento HTML**

1. **Con la componente Annotation Set Transfer[[26]](#footnote-26)**

Un meccanismo più semplice e robusto consiste nell’utilizzare l’elemento Annotation Set Transfer, fornito dal plugin Tools[[27]](#footnote-27) di GATE. L’Annotation Set Transfer consente di copiare o spostare le annotazioni (Annotation) di un nuovo set di annotazione (AnnotationSet), se essi si trovano tra l'inizio e la fine di un’annotazione di certo tipo particolare (HEAD,FOOTER,ecc. . Ad esempio, questo può essere usato quando un utente vuole solo eseguire una PR su una parte specifica di un documento, ad esempio il BODY di un documento HTML . L'utente specifica il nome dell’AnnotationSet e l'Annotation che copre la parte del documento che si propone di trasferire, e il nome del nuovo set di annotazione. Tutte le altre Annotation corrispondenti al testo compreso all’interno del tag d’interesse saranno trasferite al nuovo AnnotationSet .   
L’Annotation Set Transfer non ha <initParams>. Ha i seguenti parametri di runtime <runtimeParams>:

* **inputASName:** questo campo specifica l’AnnotationSet da cui saranno trasferite le annotazioni (copiati o spostati). Se non è specificato, nulla sarà utilizzato il set di annotazione di default.
* **outputASName:** questo campo specifica l’AnnotationSet cui le annotazioni saranno trasferite. Se è lasciato vuoto, sarà utilizzato AnnotationSet di default.
* **tagASName:** questo campo specifica l’AnnotationSet che contiene l'annotazione che copre la parte rilevante del documento da trasferire. Il valore predefinito per questo parametro è " Original markups”. Se è lasciato vuoto, sarà utilizzato il set di annotazione di default.
* **textTagName:** questo campo specifica quali annotazioni devono essere trasferite dall’AnnotationSet inputAsName. Il valore predefinito per questo parametro è 'BODY' . Se è lasciato vuoto, poi saranno trasferite tutte le annotazioni dalla nota serie inputASName. Se è trovata una delle annotazioni elencate in questo campo esse saranno trasferite . Se non è trovata alcuna annotazione, essa può essere trasferita o meno a seconda del valore fornito al campo, *copyAllUnlessFound* (vedi sotto) .
* **copyAnnotations:** questo campo specifica se le annotazioni devono essere spostate (cut) o copiate (copy). Il valore predefinito falso sarà di spostare le annotazioni, sottraendoli AnnotationSet inputASName. Se impostato a true, saranno copiate le annotazioni.
* **transferAllUnlessFound:** questo campo specifica cosa dovrebbe succedere se non è trovata alcuna annotazione di copertura. Il valore predefinito è true. In questo caso, tutte le annotazioni saranno copiate o spostate (secondo l’impostazione del parametro copyAnnotations ) se non viene trovata alcuna annotazione di copertura . Se impostato su false, nessuna annotazione sarà copiato o spostato (si consiglia di impostarlo sempre a *false* per risparmiare capacità di elaborazione).
* **annotationTypes:** lista dei nomi delle annotazioni, che saranno trasferite o copiate se e solo se saranno trovati nell’AnnotationSet dell’outputAsName.

 *Dobbiamo eseguire comunque i nostri Gazetteer e Tokeniser su tutto il documento, poiché sono risorse che non dipendono dalle annotazioni o dagli AnnotationSet.*

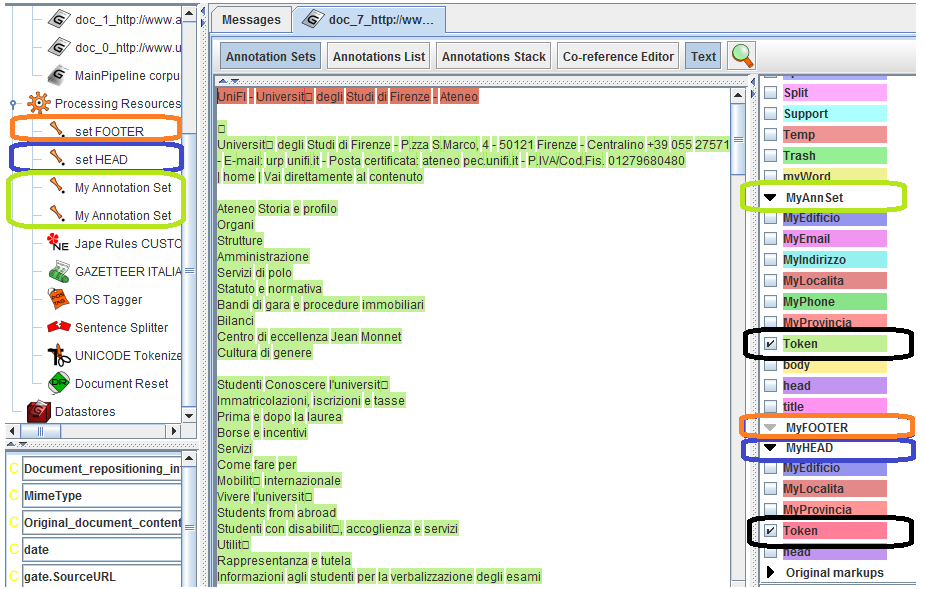
Ad esempio, supponiamo di voler eseguire le nostre regole JAPE, in un documento HTML, ma solo all’interno dell’annotazione HEAD presente di default nell’AnnotationSet Original markups. Osservate bene la fig.21 ,abbiamo creato tre AnnotationSet MyAnnSet, MyHEAD, MyFOOTER dove ha tutte e tre è stata passata l’annotazione Token di default di GATE. L’AnnotationSet MyFOOTER non fornisce alcuna annotazione, poiché nel documento HTML in esame non era presente il footer e assegnando a false il valore **copyAllUnlessFound** come abbiamo consigliato di fare, non copia l’annotazione, mentre l’AnnotationSet MyAnnSet, fornisce l’annotazione Token passatogli di default da GATE, a questo punto si può notare come nell’AnnotationSet **MyHEAD** settato con i seguenti valori:

**inputASName:MyAnnSet (AnnotationSet da cui si prendono le annotazioni)  
outputASName: MyHEAD (AnnotationSet su cui vanno a finire le annotazioni.)  
tagASName: Original markups (AnnotationSet che contiene il textTagName può non essere Original markups.)  
textTagName: head (l’annotazione di OriginalMarkups su cui lavorare, di default è tutto il documento html.)  
copyAnnotations: true (copia l’annotazione su outputas.)**

**transferAllunlessfound: false (copia l’annotazione se e solo se trovi qualcosa nel documento**.)

**annotationtypes: Token (la lista delle annotazioni da copiare - spostare.)**

Abbiamo che l’annotazione Token presente in MyAnnSet è fatta “girare” solo all’interno del tag head dell’Original markups di GATE, cioè in pratica l’annotazione Token lavora solo sull head del documento HTML.

****

**Fig.21a:Esempio applicativo dell’Annotation Set Transfer.**

*Annotation Set Transfer e Generic Tagger hanno la stessa icona ma sono due cose diverse.*

*Notare nella fig.21a che sono presenti due elementi My Annotation Set nella pipeline a sinistra questo perché volevo far mostrare che è possibile lavorare con gli Annotation Set Transfer sullo stesso set di annotazioni (AnnotationSet), tutte le volte che si vuole.*

1. **Con il componente Flexible Gazetter[[28]](#footnote-28)**

Il Flexible Gazetteer fornisce agli utenti la flessibilità di scegliere il proprio ingresso personalizzato e di un Gazetter esterno. Ad esempio, l'utente potrebbe voler sostituire parole del testo con le loro forme di base prima di eseguire il Gazetteer. Il Flexible Gazetteer esegue l’annotazione di Lookup su un documento in base ai valori di una Feature arbitraria o di una Annotation (annotazione) di tipo arbitrario, utilizzando un Gazetter esterno. È importante utilizzare un Gazetteer esterno come questo perché permette l'utilizzo di qualsiasi tipo di Gazetteer (ad esempio un Gazetteer Ontologico).   
Il Flexible Gazetter a i seguenti parametri di <Runtime>:

* **inputASName**: L’ AnnotationSet (set di annotazioni) dove il Flexible Gazetter deve cercare il tipo di *AnnotationType.Feature* specificato nel parametro di <initparam> *inputFeatureNames*. Cioè dove deve cercare quel tipo di annotazione con quel certo tipo di Feature. L’esempio classico se volessimo tracciare i valori *<div id=”elementOfGazetter”></div>* nel documento HTML potremmo inserire l’AnnotationSet di default di GATE *Original Markups* in cui è presente il tipo di annotazione “*div* “, se volessimo identificare solo div aventi la feature “id” di un certo valore presente tra gli elementi del nostro Flexible Gazetter
* **outputASName**: l’ AnnotationSet d’uscita delle annotazioni AnnotationType.Feature lette nel inputAsNAme.

Il Flexible Gazetter a i seguenti parametri di < initParams >:

* **inputFeatureNames** : quando selezionati, questi valori vengono usati per sostituire il testo originale corrispondente. Per ogni caratteristica, un documento temporaneo viene creato dai valori delle caratteristiche speci fi cato sui tipi di annotazione speci fi cato. Ad esempio: per **Token.root** il documento temporaneo avrà il contenuto di ogni Token sostituito con il suo valore di root. In caso di annotazioni dello stesso tipo in ingresso cioè una sovrapposizione, è considerato solo il valore della prima annotazione. Qui, si ricorda che l'ordine delle annotazioni è deciso utilizzando la gate.util.O ff classe setComparator.
* **gazetteerInst** : l'istanza che fa riferimento al nostro Gazetteer reale, quello che si dovrebbe eseguire su un documento temporaneo. Questo genera le annotazioni di ricerca diretta con le caratteristiche e deve essere un'istanza di gate.creole.gazetteer.Gazetteer che è già stato creato. Per farla breve il suo valore è il riferimento a un Gazetter classico di ANNIE presente precedentemente nella PIPELINE di GATE. Una volta che il Gazetter esterno ha annotato il testo con annotazioni di ricerca diretta, sul documento temporaneo si va a dare un’occhiata alle annotazioni sul documento originale. Infine, il documento temporaneo viene eliminato.

Mostriamo un’esempio dove andiamo a tracciare dal nostro AnnotationSet *MyAnnSet* in cui tramite l’Annotation Set Transfer visto precedentemente sono stati copiati e trasferiti le annotazioni “*div*”dall’AnnotationSet *Original markups* con feature “*id*” con valore presente nella lista del gazetteerInst puntata da *gazetteerInst* in un’AnnotationSet chiamato *MySpecialID.*

**inputASName: MyAnnSet (AnnotationSet contenente gli AnnotationType.Feature voluti)**

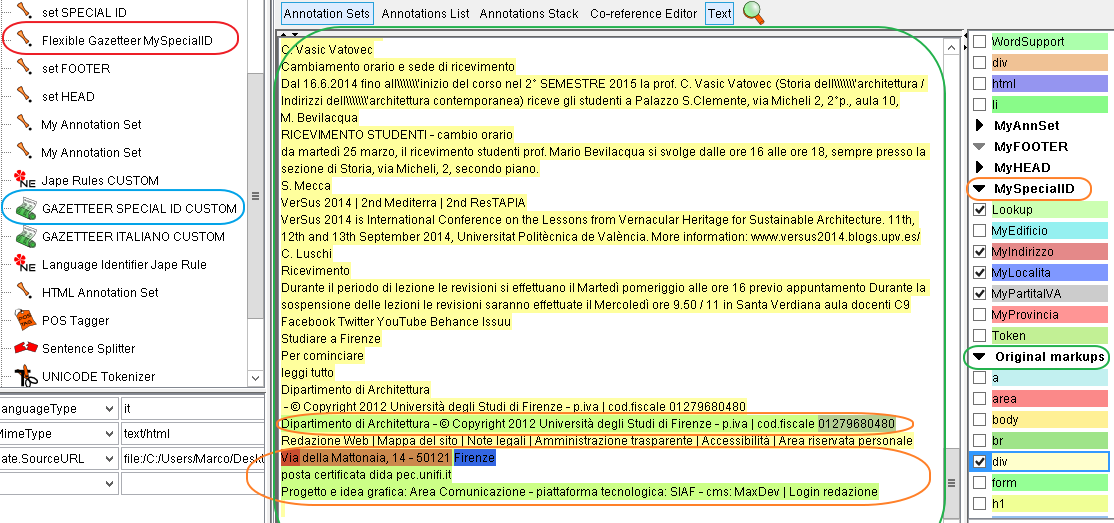
**outputASName: MySpecialID (AnnotationSet su cui vogliamo mostre gli AnnotationType.Feature volute ,**

**può essere anche lo stesso MyAnnSet)**

**inputFeatureNames: div.id (La lista degli AnnotationType.Feature da individuare nel documento)**

**gazetteerInst: GAZETTER SPECIAL ID CUSTOM (il Gazetteer esterno contenente in forma di lista tutti i**

**possibili valori delle Feature inAnnotationType.Feature).**



**Fig.21b: Esempio applicativo del Flexible Gazetter.**

Come si può notare in figura 21b il testo in giallo identifica tutti i div dle docuemnto HTML analizzato mentre il testo delll’AnnotationSet MySpecialID evidenziato in verde è costituito da tutti quei div aventi il valore del “feature” id presente nella lista del Gazetteer “GAZETTEER SPECIAL ID CUSTOM”. In figura 21c mostriamo i conteuto del file .lst dei valori degli id fa controllare e il pezzo di codice HTML in cui vi sono le informazioni:

|  |  |
| --- | --- |
| Contenuto del Gazetter che fa riferimento a un file.def che fa riferimento a una qualche lista myID.list di cui si mostra  Il contenuto di seguito | Pezzo di codice della pagina html su cui vi sono le informazioni che stiamo cercando notare che siccome il valore id “footerlinksspec” non è presente nel file.lst a sinistra esso non viene preso in considerazione nell’Annotation Set MySpecialID |
| footer  footercc  footercredits  contatti  widget\_contact\_info-3  widget-container  widget\_contact\_info  wrapper  confit-address  confit-hours  widget  widget\_text  textwidget  entry-content  su-footer-links  contenuto\_footer  contenitore\_footer  footer-nav  footercopyright  ……………………………… | <div id="footer">  <div id="footercopyright">  <h1>Dipartimento di Architettura</h1>&nbsp;-&nbsp;&copy;&nbsp;Copyright 2012 Universit&agrave; degli Studi di Firenze - p.iva | cod.fiscale 01279680480</div>  <div id="footercopyright-stampa">Dipartimento di Architettura&nbsp;-&nbsp;&copy;&nbsp;Copyright 2012 Universit&agrave; degli Studi di Firenze - p.iva | cod.fiscale 01279680480</div>  <div id="footerlinksspec">  <a href="cercachi-per-4303.html">Redazione Web</a> | <a href="mod-cmpro-showmap.html">Mappa del sito</a> |  <a href="http://www.unifi.it/cmpro-v-p-9820.html">Note legali</a> |  <a href="http://www.unifi.it/cmpro-v-p-9661.html">Amministrazione trasparente</a> |  <a href="http://www.unifi.it/cmpro-v-p-10048.html">Accessibilit&agrave;</a> | <a href="http://www.unifi.it/index.php?module=ofform2&amp;mode=3">Area riservata personale</a>  </div>  <div id="footercc">Via della Mattonaia, 14 - 50121 Firenze<br>posta certificata dida<img src="http://wwwstatic.siaf.unifi.it/global/images/ATsm.gif" alt="AT" width="11" height="10">pec.unifi.it</div>  <div id="footercredits">Progetto e idea grafica: <a href="http://www.unifi.it/cercachi/show.php?f=s&amp;codice=032000">Area Comunicazione</a>  - piattaforma tecnologica: <a href="http://www.siaf.unifi.it/">SIAF</a>  - cms: <a href="http://www.maxdev.it">MaxDev</a> | <a href="admin.php">Login redazione</a>  </div>  <div id="rigafooter">&nbsp;</div>  </div> |

**Fig.21c: Continuazione esempio applicativo del Flexible Gazetter.**

Notare che ci si potrebbe chiedere come mai non venga preso tutto il contenuto del *<div* *id=footer></div>* dopotutto è nella lista, eppure viene saltato il pezzo di codice corrsipondente a *<div* *id=* *footerlinksspec ></div>* all’interno di esso, non solo se dalla nostra lista di valori di *id* togliessimo *footercopyrigths, footercc* e *footercredits* il Flexible Gazetteer non prenderebbe niente. Infatti per come è strutturato lo strumento lui prende il testo contenuto nel div con quell’id, ma non prende il testo contenuto in altri div presenti all’interno di esso, prende solo il testo relativo a quell’id. Pensando all’ analogia della struttura ad albero di HTML si potrebbe dire, “si vede solo ciò che sta sul nostro ramo e non quello che c’è nei rami delle foglie che crescono lungo il ramo”.

Entro i limiti di questo studio la combinazione di Flexible Gazetter e Annotatin Set Transfer ci permette di fare tutto quello che si vuole con i documenti taggati in generale o di cui GATE riesce a creare il GATE Document.

Tutto le manovre e gli artifizi delle regole Jape e degli Annotation Set Transfer possono essere svolti con il linguaggio JAVA e l’utilizzo dell’API di GATE, sulla documentazione ufficiale vi sono esempi e tutorial.

***Se e solo se****, loscopo è* ***solo*** *quello di creare delle annotazioni ed estrarre il testo che esse catturano nel documento sotto forma di stringa Java ad esempio, allora la soluzione migliore (che è anche quella più semplice), è gestire la ricerca nelle varie parti del documento HTML con gli Annotation Set Transfer che sono testati e “robusti” e gestire l’estrazione del contenuto attraverso il linguaggio Java e l’API di GATE, in questo modo si è in grado di creare meccanismi di ricerca che per esempio prima consultando il set di annotazioni MyHEAD, se sono soddisfatti si fermano e vanno al prossimo documento, altrimenti vanno a utilizzare il set di annotazione MyAnnSet. Insomma se si trova l’informazione voluta nell’HEAD del documento HTML a posto, non stiamo a consultare il resto, altrimenti andiamo a guardare nel BODY del documento. Nel caso si introducano Ontologie o controlli multi lingua, il tutto diventa più complesso e sono necessarie altre osservazioni.*

1. **Identificazione della lingua del documento web**

In questo capitolo mostreremo come rendere il nostro programma più efficiente, nelle nostre pagine web avremo a che fare con solo due tipi di lingua italiano e inglese. Come abbiamo già detto GATE di default fornisce già tutti gli strumenti necessari per un’ottima analisi della lingua inglese mentre per l’italiano sono necessarie le modifiche apportate ai vari elementi come descritte nel capitolo precedente. Dobbiamo fornire al programma la possibilità di interpretare la lingua con cui è scritto il documento e la scelta di caricare o no diverse file.

**E’ utile?** Dipende da quello che vogliamo fare e se teniamo alla qualità del risultato piuttosto che alla velocità.

**E’ efficiente in termini di runtime del programma?** Il controllo della lingua naturalmente rallenta il programma ma bisogna tenere conto di tutti i vari passi che sono svolti, per esempio nel nostro studio usiamo il controllo della lingua con GATE non per velocizzare l’elaborazione svolta da GATE, *ma per fornire informazioni supplementari all’API Google Maps che velocizza il processo di ricerca* per esempio se gli dico che la regione d’interesse è in Italia, “scarta” da subito il resto del mondo.

**Per il nostro studio ha deciso che è utile,mostriamo come si realizza attraverso tre meccanismi:**

1. **Un file . gapp del tipo *gate.util.persistence.ConditionalSerialAnalyserControllerPersistence* , il “Conditional”[[29]](#footnote-29) indica che si può imporre una condizione a ogni elemento della PIPELINE ,in questo caso la lingua, e poi da esso decidere su quale delle componenti PR del punto 6.1 continuare.**
2. **Un meccanismo di controllo della lingua del documento in esame.**
3. **Una condizione if-else a livello Java per far fornire dal programma JAVA le opportune informazioni a Google Maps (lo vedremo in dettaglio nel progetto GATE Embedded in JAVA).**

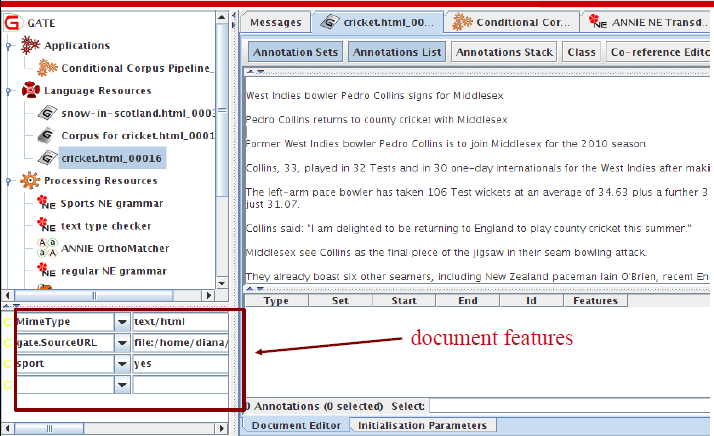
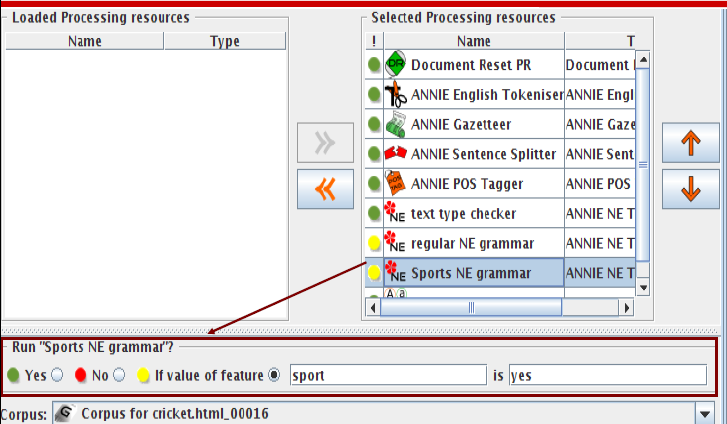
Per ora ci concentreremo sui primi due punti il terzo sarà visto meglio in dettaglio nel capitolo del progetto JAVA , abbiamo detto che è necessario un meccanismo di controllo della lingua per il documento web, GATE fornisce già due elementi , (due plugin in pratica), Textcat[[30]](#footnote-30) e Lingpipe Language Identifier[[31]](#footnote-31) ,noi utilizzeremo il secondo , senza entrare troppo in dettaglio del funzionamento, che si basa su machine learning e statistica dei modelli del linguaggio, andremo a vedere giusto come utilizzarlo.

1. **Scelta della lingua attraverso il Conditional Corpus**

In GATE, è possibile impostare una risorsa di elaborazione in applicazione (PR, Processing Resource) per l'esecuzione o meno secondo determinate circostanze. È possibile avere caricato diversi PR, e lasciare che il sistema scelga automaticamente quale far correre, per ogni documento.

Questo è molto utile quando si hanno testi in più lingue, o di tipi diversi, che potrebbero richiedere diversi

tipi di lavorazione ad esempio, se avete un misto di documenti in italiano e inglese nel corpus, si potrebbero avere qualche PR dipendente dalla lingua e alcuni che non lo sono. È possibile configurare l'applicazione per eseguire i PR pertinenti secondo il giusto documento automaticamente.



**Fig.22:Come appare nell’interfaccia GATE.**

**Come funziona**

In primo luogo dobbiamo distinguere tra i vari testi, e annotarli con valori diversi, per ogni funzione del documento:

• Utilizzare una grammatica JAPE per trovare informazioni in genere metadati che identificano la lingua usata nel documento, ad esempio, riconoscere le parole del luogo scritte nel testo da un dizionario geografico.

• La grammatica JAPE aggiunge una feature al documento "lang" con il valore "yes" o "no" ai documenti.

• Aggiungere sia la grammatica italiana sia la grammatica inglese per l’applicazione.

• Impostare la grammatica italiana da eseguire solo se il valore della funzione "lang" è "yes"

• Impostare la grammatica inglese da eseguire solo se il valore è "no”.

E’ possibile imporre una condizione di “utilizzo” a ogni singolo elemento della PIPELINE direttamente dall’interfaccia grafica GATE questo è possibile se la nostra PIPELINE creata dal nostro file . gapp/.xgapp è del tipo **Conditional[[32]](#footnote-32)** e impostare il valore della feature direttamente da esso:

**<runMode>4</runMode>**

**<featureName>language</featureName>**

**<featureValues>yes</featureValue>**

E’ possibile fare riferimento naturalmente anche a più feature basta utilizzare una lista come mostrato sotto.

**<runMode>4</runMode>**

**<featureName>**

**<gate.util.persistence.CollectionPersistence>**

**<localList>**

**<string>language</string>**

**</localList>**

**<collectionType>java.util.ArrayList</collectionType>**

**</gate.util.persistence.CollectionPersistence>**

**</featureName>**

**<featureValues>**

**<gate.util.persistence.CollectionPersistence>**

**<localList>**

**<string>yes</string>**

**</localList>**

**<collectionType>java.util.ArrayList</collectionType>**

**</gate.util.persistence.CollectionPersistence> </featureValue>**

1. **Scelta della lingua con Lingpipe Language Identifier**

Come suggerisce il nome, questo PR è utile per identificare la lingua di un documento o porzione di testo. Questa PR utilizza un file modello per identificare la lingua di un testo. Un modello è fornito in questo plugin nella sottodirectory resources/models. Il PR ha i seguenti parametri di runtime:

**annotationSetName**: Il set di annotazione utilizzato per l'input e l'output ; ignorato se AnnotationType è vuoto .

**annotationType**: Se questo è fornito, il PR classifica il testo sottostante ogni annotazione del tipo specificato e memorizza il risultato come una caratteristica su tale annotazione. Se è lasciato vuoto (null o vuoto), il PR classifica il testo di ogni documento e memorizza il risultato come una caratteristica del documento.  
**languageIdFeatureName:** Il nome della feature del documento o annotazione utilizzato per memorizzare i risultati.  
  
A differenza di molti altri PR (che producono annotazioni), questo aggiunge ad annotazioni già esistenti delle informazioni aggiuntive ai documenti attraverso le feature delle annotazioni. La precisione della classificazione del linguaggio aumenta su lunghe campate di testo (paragrafi piuttosto che frasi).

Se forniamo i seguenti valori all’elemento LLI.

**annotationSetName = Original Markups**

**annotationType = head**

**languageIdFeatureName = languagePipeType**

esaminerà il testo del documento contenuto nel tag head dell’AnnotationSet di default di GATE “Original markups”, fornendo il risultato dell’elaborazione con una Feature di nome languagePipeType che identifica il tipo di lingua usato con l’acronimo it,en,fr,ecc.

*Per i fogli HTML è molto utile perché è possibile parsare in modo differente diversi parti del documento*, non andremo a un tale livello di dettaglio, non è lo scopo principale del nostro studio noi ci limiteremo a controllare l’intero foglio HTML (più è lungo il testo analizzato dall’identificatore maggiore, è la possibilità di prendere la lingua giusta) del documento, se è restituito il valore *it,* caricheremo il file. gapp per la lingua italiana se è restituito *en* il file.gapp per la lingua inglese.

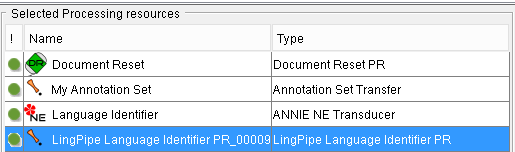
1. **Scelta della lingua con le Feature del documento HTML**

Nella quasi totalità i documenti HTML (non è detto che i file PDF e Word abbiano tale informazione nei loro metadati), forniscono **la feature lang** **all’inizio del documento html**, essa è un’informazione standard che identifica il tipo di linguaggio utilizzato nel documento HTML corrente, quindi prima di consultare il risultato del Lingpipe Language Identifier, consulteremo tale valore, semplicemente perché se il creatore del documento html ha specificato la lingua del suo documento, è sicuro che quella sia la lingua su cui dovremo lavorare.

*Notare che GATE carica già di default tale Feature attraverso la coppia, lang =”xx” per ogni documento html e per ogni documento PDF o simili che l’hanno come meta informazione, siccome il nostro studio si basa su documenti web di siti universitari e simili (quindi professionali), è probabile che non andremo mai utilizzare il risultato della Lingpipe Language Identifier che è lasciato come ultima risorsa nel caso più unico che raro, di un documento che non ha tale informazione.*

1. **Le modalità di scelta della lingua**

Mostriamo un esempio utilizzando una PIPELINE di quattro PR, su cui si va a creare tre differenti Feature nella medesima annotazione per il tracciamento della lingua del testo del documento.

1. Resetta il Documento
2. Prende le feature *lang* del tag html e le mette nel

nostro annotation Set di default

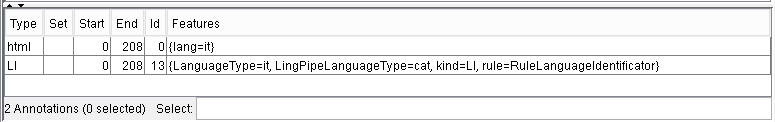
1. Grammatica JAPE per creare l’annotazione

*LanguageType* con la feature lang di default

di GATE

1. LingPipe Language Identififer,crea la feature

*LingPipeLanguageType* che identifica la lingua.



**Fig.23: Gli elementi del file .gapp utilizzati per settare le feature di identificazione del linguaggio**

**La prima feature {lang = it}:** è la feature che GATE individua di default con il HTML lang (inclusa anche come standard dal W3C), che è presa in automatico da GATE senza che sia richiesto in modo esplicito dall’utente.

**La seconda feature {LanguageType = it}:** è la feature identificata da una regola JAPE, fa la stessa cosa della prima feature solo non in automatico, ma su richiesta di determinate condizioni dell’utente, per esempio se volessimo assegnare al riferimento della feature lang non “it”, ma “italiano” questo è un buon metodo.

**La terza feature {LingPipeLanguage = cat}:** è la feature fornita dall’elaborazione del Lingpipe Language Identifier, in questo caso il documento HTML in esame aveva del contenuto sia inglese sia italiano e il Language Identifier non è riuscito a scegliere quale lingua restituire fornendo il risultato “cat”, che può essere tradotto come “non lo so”, ma è molto utile invece se vogliamo andare a elaborare singole parti del documento HTML, come FOOTER, HEAD, ecc.

**Phase: languageIdentificator**

**Input: Token Lookup html**

**Options: control = appelt**

**/\*\*Se il feature lang del documento html è it\*/**

**Rule: RuleLanguageIdentificator**

**Priority: 20**

**( ({html.lang == "it" }) ):li -->**

**{**

**gate.AnnotationSet matchedAnns=(gate.AnnotationSet)bindings.get( "li" );**

**if(matchedAnns != null && matchedAnns.size() > 0) {**

**doc.getFeatures().put( "LanguageType" , "it");**

**outputAS.add(matchedAnns.firstNode(), matchedAnns.lastNode(), "LI", newFeatures);**

**}else{**

**doc.getFeatures().put( "LanguageType" , null);**

**outputAS.add(matchedAnns.firstNode(), matchedAnns.lastNode(), "LI" , newFeatures);**

**}**

**}**

**/\*\*Se il feature lang del documento html è en\*/**

**Rule: RuleLanguageIdentificator2**

**Priority: 20**

**( ({html.lang == "en" }) ):li2 -->**

**{**

**gate.AnnotationSet matchedAnns=(gate.AnnotationSet)bindings.get( "li2" );**

**if(matchedAnns != null && matchedAnns.size() > 0) {**

**doc.getFeatures().put( "LanguageType" , "en" );**

**outputAS.add(matchedAnns.firstNode(), matchedAnns.lastNode(), "LI", newFeatures);**

**}else{**

**doc.getFeatures().put( "LanguageType" , null);**

**outputAS.add(matchedAnns.firstNode(), matchedAnns.lastNode(), "LI" , newFeatures);**

**}**

**}**

**/\*\*Se il feature lang del documento html non nè it nè en o non è un documento html(e.g. pdf)\*/**

**Rule: RuleLanguageIdentificator3**

**Priority: 20**

**( ({html.lang != "it" ,html.lang != "en" }) ):li3 -->**

**{**

**gate.AnnotationSet matchedAnns=(gate.AnnotationSet)bindings.get( "li3" );**

**if(matchedAnns != null && matchedAnns.size() > 0) {**

**doc.getFeatures().put( "LanguageType" ,”en”);**

**outputAS.add(matchedAnns.firstNode(), matchedAnns.lastNode(), "LI", newFeatures);**

**}else{**

**doc.getFeatures().put( "LanguageType" ,”en”);**

**outputAS.add(matchedAnns.firstNode(), matchedAnns.lastNode(), "LI" , newFeatures);**

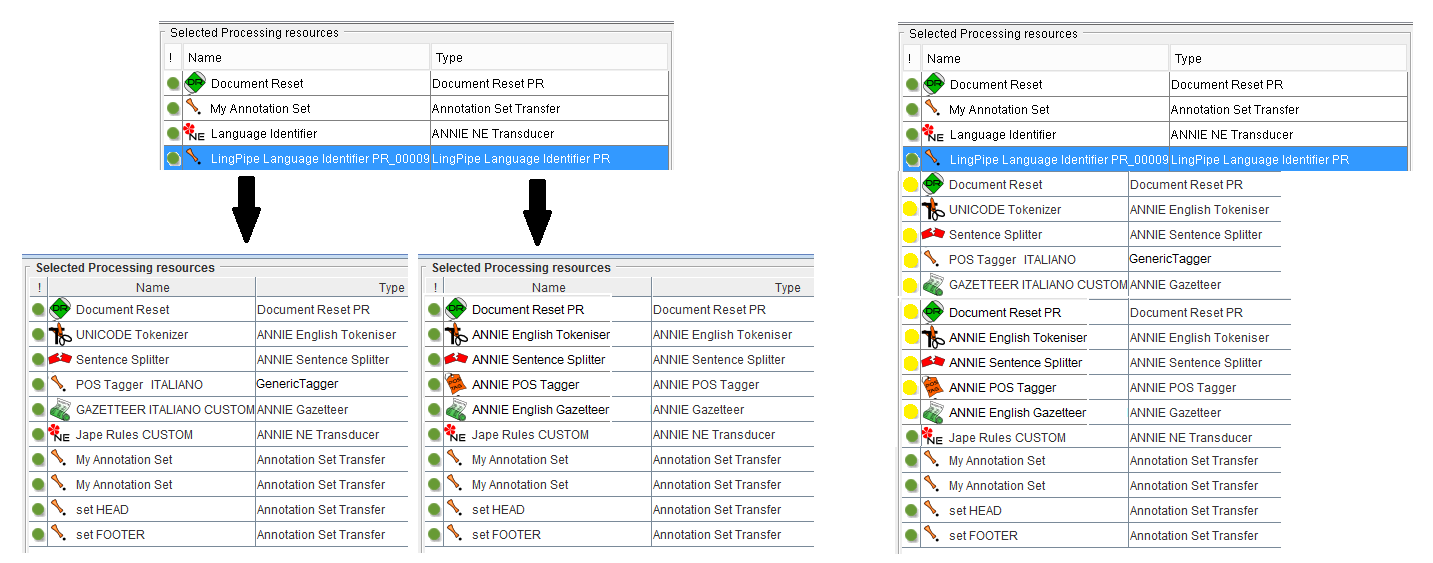
**}**

**}**

**Fig.24: Regola per la cattura della feature lang di default di GATE sui documenti HTML e non.**

1. **Conclusione sulla modalità di scelta della lingua per questo studio**

A questo punto possiamo o prendere tale valore delle feature con il programma JAVA e scegliere il file.gapp inerente alla lingua identificata dalla sigla “xx” oppure impostare un unico file .gapp settato con i giusti valori condizionali.

****

**Fig.25:Le due modalità per integrare la lingua nei documenti o con più file.gapp(sinistra) o con un unico file .gapp(destra)**

*Notare che il “marker” giallo nella seconda modalità identifica i PR cui è stata assegnata una condizione, mentre il pallino verde identifica quelli che vengo “sempre” caricati, quindi le regole Jape e i Set Annotation Transfer sono comuni a tutte le lingue almeno nel nostro studio.*

**Quale scegliere?**

**La prima modalità:** è più conveniente, quando si vuole lavorare *su più lingue***,** la seconda modalità in questo caso funziona sempre, ma avrebbe lo svantaggio di caricare un numero non indifferente di elementi che non sarebbero fatti caricare comunque e che pesano nel complessivo, senza contare che se *volessimo far lavorare la condizione su più di una feature* questo non è possibile nella seconda modalità. La prima modalità, infatti, permette di gestire la condizione a livello JAVA quindi a un livello superiore, lo svantaggio è abbastanza evidente è dovuto al fatto che è necessario fare girare su uno stesso documento due file.gapp e che bisogna realizzare due differenti Corpus di documenti in due diversi momenti quindi peso sul processore e runtime più lungo.

**Seconda Modalità:** nel caso in cui si lavori *su due o tre sole lingue* si è propensi a utilizzare la modalità con unico file .gapp che risulta sempre più veloce di un codice con confronto condizionale in java, tale meccanismo è poi perfettamente integrata in GATE ed è automatica.

**Terza Modalità:** nel capitolo 7,durante la costruzione del progetto Java, viene descritta l’API Java di Apache Tika come uno strumento per l’identificazione del linguaggio e valida alternativa alle precedenti due.

*La prima e seconda modalità sono uguali a livello concettuale la scelta va al progettista, è possibile togliersi ogni dubbio in merito con una verifica su un set di documenti e sulla velocità di runtime per decidere quale la migliore scelta su tot. Lingue e s u tot. Feature del documento***.**

**Poiché nel nostro studio diamo grande importanza anche alla velocità di esecuzione del programma e poiché le lingue utilizzate sono solo due inglesi e italiano, la seconda modalità è la soluzione migliore.**

**Come decidiamo quale meccanismo per l’identificazione del linguaggio utilizzare?**

Abbiamo definito tre feature per identificare linguaggio:

1. **Lang:** più veloce ma per i file non html ci restituisce un valore che non è immediatamente gestibile, per esempio un file pdf restituisce null che non è una stringa e che potrebbe far lanciare eccezione al programma escludendo che tutto questo è gestibile, se abbiamo per il 90%, solo documenti pdf, doc, ecc. difficile ma possibile), questo rallenta troppo il programma che fa un sacco di controlli inutili.
2. **LanguageType:** prende lo stesso valore individuato dalla feature lang ma lo fa attraverso una regola JAPE che possiamo gestire a piacimento, l’elaborazione della regola JAPE è l’unica differenza di velocità tra le due.
3. **LanguagePipeType:** è la feature più lenta individuata dal Lingpipe Language Identifier ed è forse la più potente di tutte, anche se a volte per documenti scritti in più lingue, non individua la giusta lingua di riferimento, purtroppo per garantire un risultato buono dell’individuazione della lingua necessità di “leggere” grosse porzioni di testo .

*Il nostro compromesso finale è individuare la lingua del documento Web con un aumento del tempo di runtime abbastanza trascurabile, attraverso l’individuazione della feature lang nel tag html per i soli documenti HTML attraverso una nostra regola JAPE (quindi la Feature LanguageType in precedenza descritta), utilizzando di default per le pagine web non html il parser della lingua inglese, questo per evitare di perdere tempo ad analizzare pagine web di grosse dimensioni che rendono obsoleto il guadagno di tempo fornito dall’identificazione della lingua nella ricerca con Google Maps usato in questo studio.*

1. **Progetto JAVA: EstrazioneDatiWebPageToRecordSQLWithGATE**

**Cosa abbiamo fatto finora:**

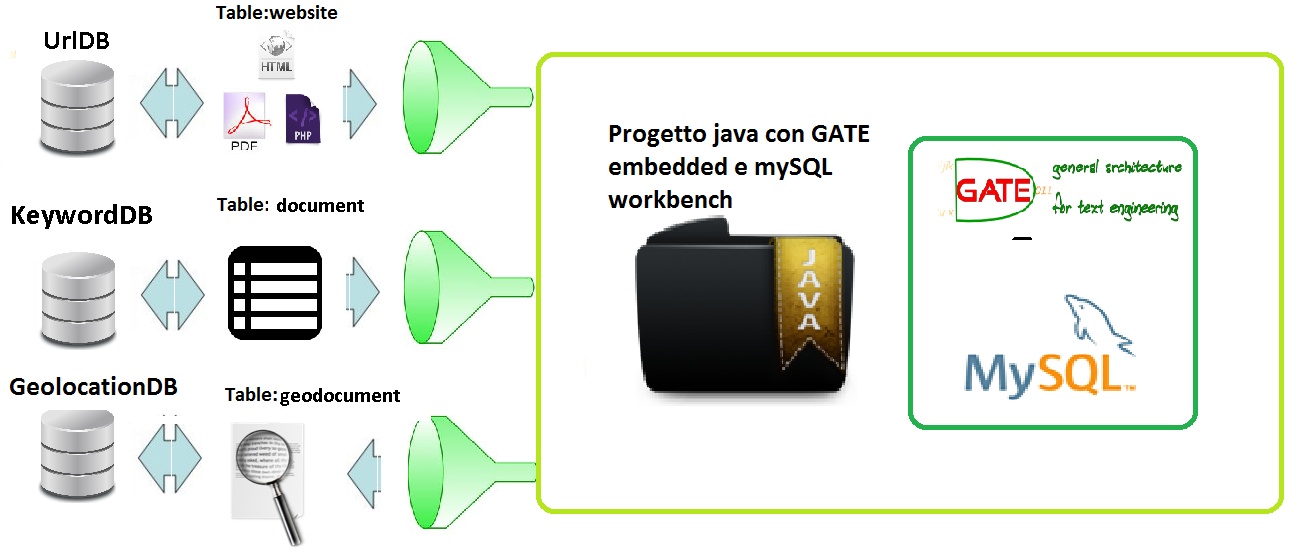
* Abbiamo definito i concetti teorici sul web semantico e i problemi dell’informazione strutturata nella rete.
* Abbiamo descritto il software GATE in dettaglio.
* Abbiamo definito i vari elementi della PIPELINE che utilizzeremo.
* Abbiamo mostrato come utilizzare al meglio i tag dei documenti HTML con le annotazioni di GATE.
* Abbiamo definito un “sistema” per l’identificazione della lingua nel documento Web in esame.

**Vediamo quali azioni ci rimane da svolgere con il nostro programma JAVA:**

1. **Prendere gli URL dei documenti Web del database UrlDB nella tabella Website del sistema OntoStar.**
2. **Creare per ognuno di questi URL un GATE Document attraverso CORPORA come descritto nella documentazione ufficiale di GATE.**
3. **Impacchettare i GATE Document in un Corpus.**
4. **Far girare sul tale Corpus la nostra/e Pipeline costruito/e con il/i relativo/i file .gapp e le annotazioni semantiche e non, da esse create e integrate con un controllo della lingua come quello in precedenza accennato.**
5. **Estrazione del contenuto delle annotazioni semantiche sotto forma di stringa JAVA e gestione delle medesime per ottimizzare la geo localizzazione.**
6. **Creazione di corretti oggetti java GeoDocument che strutturano i contenuti delle annotazioni estratte.**
7. **Elaborazione della latitudine e della longitudine della località descritta nei campi del GeoDocument con l’ausilio dell’API Google Maps.**
8. **Inserimento delle coordinate di Google Maps nei corretti oggetti JAVA GeoDocument.**
9. **Integrazione delle informazioni presenti nei nostri GeoDocument con le informazioni già presenti nei Document del Sistema OntoStar, cercando di colmare i campi vuoti (null) della tabella GeoDocument se possibile e avere un successivo confronto con i risultati di tale progetto.**
10. **Integrazione con il motore di ricerca SearchMonkey**
11. **Inserimento dei GeoDocument in una lista dove ogni elemento della lista è il risultato della geo localizzazione estratto da un documento GATE del Corpus.**
12. **Inserire la lista di GeoDocument come record nella tabella GeoDocument del database GeolocationDB.**

**Stabiliamo le path del sistema operativo e importiamo i file JAR necessari nel nostro programma** per utilizzare L’API GATE Embedded**[[33]](#footnote-33)** è un framework orientato agli oggetti (o libreria di classi) implementato in Java. E ' utilizzato in tutti i sistemi basati su GATE e fornisce tutti gli elementi di base per programmare con GATE. Per utilizzare l’API di GATE EMBEDDED è necessario seguire i seguenti passi:

1. Aggiungere $ GATE\_HOME / bin / gate.jar e i file JAR in $ GATE\_HOME / lib al CLASSPATH di Java ($ GATE\_HOME è la directory principale GATE).
2. Dire a Java che il kit GATE Unicode è un'estensione:-Djava.ext.dirs = $ GATE\_HOME / lib / ext NB (cioè è necessario solo per le applicazioni GUI che devono sostenere l'immissione di testo Unicode);
3. Altre applicazioni come da linea di comando o applicazioni web in genere non hanno bisogno di GUK.
4. Copiare nella directory del progetto Java il file log4j.properties contenuto nella cartella bin di GATE.
5. Per inizializzare GATE all’interno del programma JAVA basta utilizzare il comando **Gate.init();**



**Fig.26: Immagine esplicativa di ciò che fa il nostro programma Java.**

**Prima di continuare forniamo alcune definizioni sugli elementi di base utilizzati dall’API di GATE:**

* **Document**, documento del web (HTML, PDF, DOC, ecc.) che è convertito nella sua sola parte testo opportunamente taggato.
* **Corpus**, un contenitore di tutti i GATE Document che dobbiamo analizzare.
* **CorpusController**, imposta un’opportuna pipeline (anche tramite file.gapp) per un determinato Corpus.
* **FeatureMap**, una mappa delle Feature dello specifico documento GATE (informazioni aggiuntive).
* **Feature**, l’informazione aggiuntiva a una determinata Annotazione Semantica (Annotation).
* **Annotation**, l’annotazione base di GATE in precedenza discussa.
* **AnnotationSet**, un set di annotazioni che lavorano per esempio nella stessa parte di testo del documento.
* **CorpusController,** applica la PIPELINE del nostro studio (creata dal file.gapp) sul Corpus del progetto, contengono i vari PR in precedenza descritti ANNIE, Lingpipe, Tools, ecc. E’ in pratica il nostro elemento per le operazioni d’inizializzazione, settaggio e run di GATE.

1. **La nostra prima classe Java**

Come base di partenza si parte dalla classe Java fornita dalla documentazione ufficiale di GATE **StandAloneANNIE.java[[34]](#footnote-34)** e la adattiamo ai nostri scopi, creiamo quindi la nostra prima classe Java **EstrazioneDatiWithGATE.java.** Dopo aver importato gli opportuni file .jar di GATE nel nostro programma come accennato in precedenza, s’inizializza GATE con le impostazioni di default e gli si fa caricare in automatico il nostro file .gapp/.xgapp. Cominciamo con il codice per inizializzare GATE nella classe MainGate.java con il metodo main(). Come mostrato in *fig.27* ci sono dei passi da seguire:

1. **Gate.init():** come accennato, in precedenza, richiama l’API GATE con tutti i parametri di default.
2. **gate.init():** invocail metodo init() del CorpusController che carica per GATE il nostro file .gapp/.xgapp con la rispettiva PIPELINE di GATE .
3. **gate.setCorpus():** invoca il metodo setCorpus() del CorpusController che setta il corpus per GATE, più avanti mostreremo come sarà creato tale Corpus.
4. **gate.execute():** invoca il metodo execute() del CorpusController e si occupa di caricare la PIPELINE di GATE sul Corpus.
5. ***MainFrame.getInstance().setVisible(true):*** E’ possibile anche lavorare direttamente dall’interfaccia grafica di GATE, con la semplice riga di codice.

**private static CorpusController controller;**

**/\*\* Tell GATE's controller about the corpus you want to run on.**

**\* @param corpus il corpus da settare \*/**

**public void setCorpus(Corpus corpus) {controller.setCorpus(corpus);} // setCorpus**

**/\*\* Run GATE. \*/**

**public void execute() throws GateException {controller.execute();} // execute()**

**/\*\* Initialise the GATE system. This creates a "corpus pipeline" application that can be used to run sets of documents through the extraction system. \*/**

**public void init() throws GateException, IOException {**

**Out.prln( "Loading file .gapp...");**

**//Carica tutte le applicazioni della GATE\_HOME attraverso il file .gapp**

**File home = Gate.getGateHome();**

**File gapp = new File(home, "custom/gapp/geoLocationPipeline.xgapp");**

**controller = (CorpusController) PersistenceManager.loadObjectFromFile(gapp);**

**Out.prln("...file .gapp loaded " );**

**} // init ()**

/\*\*Run from the command-line, with a list of URLs as argument.

**\*This code will run with all the documents in memory - if you want to unload each from memory after use, add code to store the corpus in a DataStore.\*/**

**public static void main(String args[]) throws InterruptedException, InvocationTargetException{**

**EventQueue.invokeLater(new Runnable() {**

**public void run() {**

**try{**

**Out.prln( "Initialising GATE..." );**

**Gate.init();**

**Out.prln( "...GATE initialised" );**

**// Inizializza GATE(this may take several minutes)**

**MainGate gate = new MainGate();**

**gate.init();**

**// Con questa linea di codice vediamo tutto l'output sulla interfaccia GATE**

**MainFrame.getInstance().setVisible(true);**

**Err.prln( "...START" );**

**// Crea un Corpus e dagli un Controller (CorpusController)**

**Corpus corpus = ………….;**

**Out.prln( "Fine caricamento della PIPELINE" );**

**//settiamo il corpus**

**gate.setCorpus(corpus);**

**//carichiamo i documenti del corpus sulla nostra pipeline**

**Out.prln( "Running PIPELINE..." );**

**gate.execute();**

**Out.prln( "...PIPELINE complete" );**

**} catch (Exception e1) {**

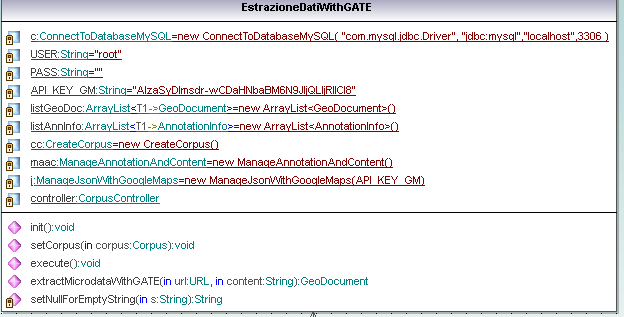
**Err.prln( "ECCEZIONE DI QUALCHE TIPO CAUSATA DA GATE" );**

**e1.printStackTrace();**

**}**

**} //run }); //runnable } //main**

**Fig.27:** **Pezzo di codice della classe EstrazioneDatiWithGATE.java solo la parte riguradante la chiamata a GATE del punto 7.1**



**Fig.28: Diagramm UML punto 7.1,La nostra prima classe Java.**

1. **Il file .gapp/.xgapp**

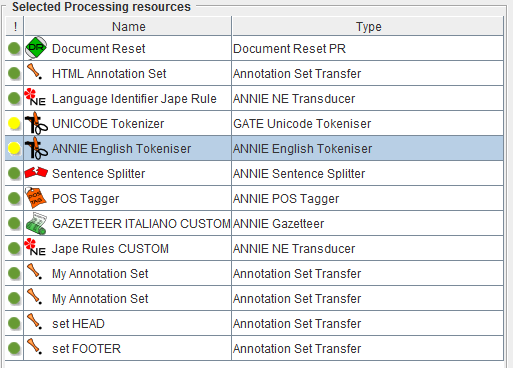
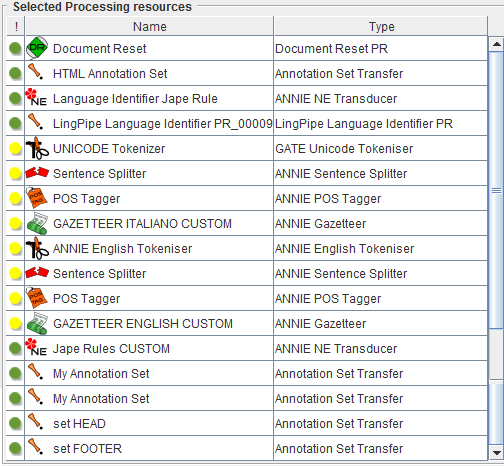
Nella nostra classe MainGate.java abbiamo creato un metodo init(), il cui scopo è di caricare sull’API GATE del programma il nostro file .gapp/.xgapp, in pratica è un file XML che costruisce una “pila” di elementi PR di GATE come ANNIE, LingPipe descritti in precedenza, realizzando ogni singolo elemento PR attraverso la loro versione XML, nella documentazione ufficiale si trovano esempi e descrizioni su come convertire ogni tipo di PR supportato da GATE.

Per costruire una personale PIPELINE con riferimento alle proprie liste, grammatiche e POS Tagger customizzati[[35]](#footnote-35) è necessario modificare i vari campi XML dei vari PR usati in questo studio, forniamo una descrizione dettagliata della customizzazione dei campi nei principali PR di ANNIE nell’APPENDICE D.

In alternativa si può scrivere il tutto, anche a livello JAVA utilizzando l’API di GATE, a livello concettuale non cambia niente , ma il file.gapp permette modifiche più veloci e frequenti, senza stare a modificare di molto il nostro codice e personalmente abbiamo notato che è leggermente più veloce nel caricare gli elementi PR rispetto alla sua versione in JAVA.

Per rendere la nostra PIPELINE più veloce nell’elaborazione delle pagine web abbiamo:

* Unificato le liste italiane e inglesi con un unico file riassuntivo .def quindi utilizzando un solo Gazetteer.
* Utilizzato il POS TAGGER di default ANNIE per entrambe le lingue italiano e inglese.
* Utilizzato il Sentence Splitter di default ANNIE per entrambe le lingue italiano e inglese.
* Programmato le nostre regole JAPE in modo tale che siano efficienti indipendentemente dal linguaggio, le vedremo nel prossimo paragrafo.
* Evitato il controllo d’identificazione della lingua con LingPipe Language Identifier.



**Fig.29: La PIPELINE del nostro studio costruita dal file geolocationPipeline.xgapp caricata dal metodo init() di MaingGate.java, a sinistra la versione più lenta con un controllo qualitativo della lingua, a destra con un controllo banale della lingua ma sufficiente a ricavare un guadagno efficace con l’API Google Maps.**

1. **Le regole JAPE**

Le regole Jape realizzate in questo studio, escludendo quelle di preprocessing che richiamano le liste del Gazetteer utilizzate, permettono di identificare varie informazioni all’interno del documento di testo per la geo localizzazione, mostreremo brevemente la struttura di quelle usate in questo studio.

***Regione, provincia, località***sono semplici regole Jape con riferimento diretto a delle liste:

**Phase: myGeolocation**

**Input: Lookup SpaceToken Token Sentence Split Location Person Date Organization Address Money Percent Token JobTitle Lookup: FirstPerson Name geo regioni provincie**

**Options: control = appelt**

**Rule: RuleMyRegione**

**Priority: 50**

**(**

**({Lookup.majorType==regioni})**

**):myRegione**

**-->**

**:myRegione.MyRegione = {kind = "myRegione", rule = "RuleMyRegione"}**

**Fig.30: La regola per tracciare le regioni durante la fase di myGeolocation**

**Iva, telefono ed email**, sono le regole non essenziali ma utili, utilizzano sempre delle liste corrette del gazetteer, ma utilizzano più regole Jape che inseriscono il risultato nella stessa annotazione, (un’annotazione può fare riferimento a più regole Jape che lavorano anche in fasi diverse, guardare la documentazione ufficiale per più informazioni). Tuttavia esse prendono come risultato una parte del pattern della regola identificata, nel senso che non ci interessa *email: marco@gmail.com,* ma solo *marco@gmail.com* anche se la parola *email:* è utile per individuare le informazioni nel documento:

**Rule: myEMAIL\_ADDRESS2**

**Priority: 20**

**(**

**(**

**({Token.String== "email" }|{Lookup.majorType==email})**

**({Split}|{SpaceToken}|{Token.String==":"})[0,3]**

**)[0,1]**

**(EMAIL):myEmaily**

**):myEmail2 -->**

**:myEmaily.MyEmail = {kind = "myEmaily", rule = myEMAIL\_ADDRESS2}**

**Fig.31:La regola per tracciare una parte del pattern della regola myEMAIL\_ADDRESS2 ci sono più regole per questo.**

**Indirizzo e edificio** sono invece le regole principali sono quelle che forniscono le informazioni sulla geo localizzazione del soggetto della pagina web in particolare l’indirizzo e l’edificio o azienda cui il testo del documento web sta facendo riferimento, vediamo il codice:

**Imports: {**

**import static gate.Utils.\*;**

**}**

**Phase: Indirizzo**

**Input: Token Lookup Split SpaceToken localita indirizzo provincie MyProvincia regioni industriale azienda assicurazione**

**articoloIndeterminativo articoloDeterminativo preposizioni avverbiDiLuogo locuzioni congiunzione trash conjunction**

**Options: control=all**

**MACRO:CONJ\_ITA(**

**(**

**{Lookup.majorType==articoloIndeterminativo} /\*(1)un una uno un'\*/**

**|{Lookup.majorType==articoloDeterminativo} /\*(2)il lo l' la i gli le\*/**

**|{Lookup.majorType==preposizioni} /\*(3)a agli ai al all' alla alle allo assieme attraverso \*/**

**|{Lookup.majorType==avverbiDiLuogo} /\*(4)altrove ci codesto costà costì dappertutto davanti dentro\*/**

**|{Lookup.majorType==locuzioni} /\*(5)accanto a vicino a davanti a innanzi a lontano da invece di insieme con\*/**

**|{Lookup.majorType==congiunzione} /\*(6)altresì ancora anzi ciò nonostante\*/**

**|{Lookup.majorType==conjunction} /\*(7)or and otherwise while\*/**

**)**

**)**

**MACRO:TRASH(**

**({Split}|{Lookup.majorType==trash})**

**)**

**MACRO:CONJ\_UNIVERSAL\_ITA(**

**({SpaceToken}|{Token,Token.length==1})**

**(**

**((TRASH)|(CONJ\_ITA))**

**({SpaceToken}|{Token,Token.length==1})**

**)[0,1]**

**)**

**/\*\*scala A interno 4\*/**

**Rule: RuleMyIndirizzo0**

**Priority: 15**

**(**

**({Lookup.majorType == indirizzo})**

**((CONJ\_UNIVERSAL\_ITA))[1,3]**

**({Token.kind==number,Token.length==1}|{Token.kind==number,Token.length==2}|{Token.kind==word,Token.length==1}|{Token.kind==word,Token.length==2})**

**({Lookup.majorType == indirizzo})**

**({Token.kind==number,Token.length==1}|{Token.kind==number,Token.length==2})**

**):myIndirizzo0**

**-->**

**:myIndirizzo0.MyIndirizzo = {kind = "myIndirizzo0", rule = "RuleMyIndirizzo0"}**

**/\*\*84011 AMALFI SA\*/**

**Rule: RuleMyIndirizzo1**

**Priority: 15**

**(**

**({Token.kind == number,Token.length==5})**

**/\*((TRASH)|(CONJ\_ITA))[1,3]\*/**

**(CONJ\_UNIVERSAL\_ITA)[1,3]**

**({Lookup.majorType==provincie})**

**(**

**/\*((TRASH)|(CONJ\_ITA))[1,3]\*/**

**(CONJ\_UNIVERSAL\_ITA)[1,3]**

**({Token.kind==word,Token.length==2,Token.orth==upperInitial})**

**)[0,1]**

**):myIndirizzo1**

**-->**

**:myIndirizzo1.MyIndirizzo = {kind = "myIndirizzo1", rule = RuleMyIndirizzo1}**

**/\*\*via Gioachino Rossini 40c - P.zza S.Marco 4\*/**

**Rule: RuleMyIndirizzo2**

**Priority: 15**

**(**

**({Lookup.majorType == indirizzo})**

**(**

**((CONJ\_UNIVERSAL\_ITA))[1,3]**

**({Token.kind==word,Token.orth==upperInitial})**

**)[1,5]**

**(**

**((CONJ\_UNIVERSAL\_ITA))[1,3]**

**({Token.kind==number,Token.length==2}|{Token.kind==number,Token.length==1})**

**(**

**(CONJ\_UNIVERSAL\_ITA)[0,1]**

**({Token.kind==word,Token.length==1})**

**)[0,1]**

**)[0,1]**

**/\*\*84011 AMALFI SA\*/**

**(CONJ\_UNIVERSAL\_ITA)[1,3]**

**(**

**({Token.kind == number,Token.length==5})[0,1]**

**(CONJ\_UNIVERSAL\_ITA)[1,3]**

**)[0,1]**

**(**

**({Lookup.majorType==localita}|{Lookup.majorType==provincie})**

**(CONJ\_UNIVERSAL\_ITA)[1,3]**

**)[0,1]**

**(**

**({Token.kind==word,Token.length==2,Token.orth==upperInitial})**

**)[0,1]**

**):myIndirizzo2**

**-->**

**:myIndirizzo2.MyIndirizzo = {kind = "myIndirizzo2", rule = "RuleMyIndirizzo2"}**

**/\*\*Latitudine: N 41.89547\*/**

**Rule: RuleMyIndirizzo4**

**Priority: 15**

**(**

**({Lookup.majorType == indirizzo})**

**(**

**(**

**(CONJ\_UNIVERSAL\_ITA)[1,1]**

**({Token.kind==word,Token.orth==upperInitial,Token.length==1})**

**)[0,1]**

**(**

**(CONJ\_UNIVERSAL\_ITA)[1,1]**

**({Token.kind==number,Token.length==2})**

**({Token.string=="."})**

**({Token.kind==number})**

**)**

**)[1,2]**

**):myIndirizzo4**

**-->**

**:myIndirizzo4.MyIndirizzo = {kind = "myIndirizzo4", rule = "RuleMyIndirizzo4"}**

**Rule: StreetName1**

**/\*Regola di default di gate per il tracciamento delle strade\*/**

**(**

**({Token.kind == number}**

**({Token.string == ","})?**

**)?**

**{Token.orth == upperInitial}**

**{Lookup.minorType == "street"}**

**)**

**:streetAddress -->**

**:streetAddress.MyIndirizzo = {kind = "streetAddress", rule = StreetName1}**

**Rule: POBoxAddress**

**/\*Regole di default di gate per il tracciamento della casella postale\*/**

**(**

**(({Token.string == "P"}**

**({Token.string == "."})?**

**{Token.string == "O"}**

**({Token.string == "."})?**

**) |**

**({Token.string == "PO"})**

**)**

**{Token.string == "Box"}**

**{Token.kind == number}**

**)**

**:poBox -->**

**:poBox.MyIndirizzo = {kind = "poBox", rule = POBoxAddress}**

**MACRO: EDIFICIOSUPPORT(**

**(**

**{Lookup.majorType == provincie}|{Lookup.majorType == regioni}|{Lookup.majorType == localita}|**

**{Lookup.majorType == azienda}|{Lookup.majorType == industriale}|{Lookup.majorType == assicurazione}|**

**{Token.kind==word,Token.orth==upperInitial}|{Lookup.majorType == universita}|{Lookup.majorType == edificio}**

**)**

**)**

**/\*UniFI - Universit� degli Studi di Firenze - Ateneo\*/**

**Rule: RuleEdificio**

**(**

**(**

**(EDIFICIOSUPPORT)**

**(CONJ\_UNIVERSAL\_ITA)[1,4]**

**)[0,4]**

**({Lookup.majorType == edificio}|{Lookup.majorType == universita}) /\*Universita\*/**

**(**

**(CONJ\_UNIVERSAL\_ITA)[1,4]**

**(EDIFICIOSUPPORT)**

**)[0,6]**

**):myEdificio**

**-->**

**:myEdificio.MyEdificio = {kind = "myEdificio", rule = "RuleEdificio"}**

**Fig.32: Le regole Jape principali che lavorano nella stessa fase per il tracciamento dell’indirizzo**

Le nostre regole Jape non variano secondo la lingua usata, è sufficiente cambiare il riferimento alle liste utilizzate, visto che la lingua italiana e inglese sono sufficientemente simili, i pattern Jape sono abbastanza generalizzati da essere valide per entrambe le lingue.

*Il codice completo delle regole JAPE utilizzate sara messo disponibile online o su richiesta (inviatemi una email).*

1. **Struttura del progetto**

Prima di descrivere ilfunzionamento del progetto mostriamo velocemente come è strutturato il programma:

src --|-- home --|-- estrazioneDatiGeoDocumentPerElaborato 🡪 estrazioneDatiGeoDocumentPerElaborato.java

| |-- estrazioneDatiGeoDomainDocumentPerElaborato 🡪

| | estrazioneDatiGeoDomainDocumentPerElaborato.java

| |-- estrazioneDatiInfoDocumentPerElaborato 🡪 estrazioneDatiInfoDocumentPerElaborato.java

| |-- object

| |-- utils -- | -- estrattori 🡪 classi inerenti agli estrattori utilizzati nel progetto

| | -- gate 🡪 classi inerenti alll’utlizzo di GATE

| | -- mysql 🡪 classi che si occupano della comunicazione con il databaseMySQL

| | -- setInfoParameterIta 🡪 classi di supporto per una migliore qualità del risultato

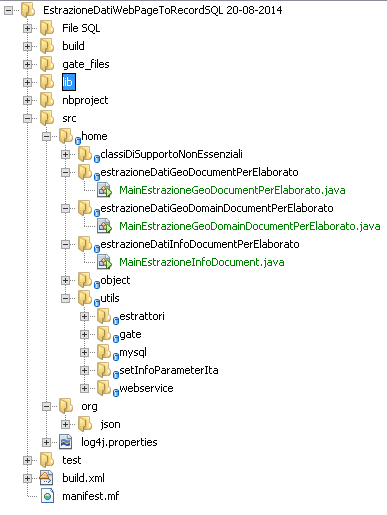
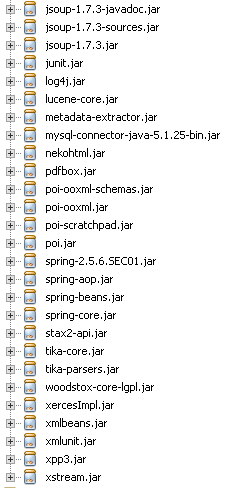
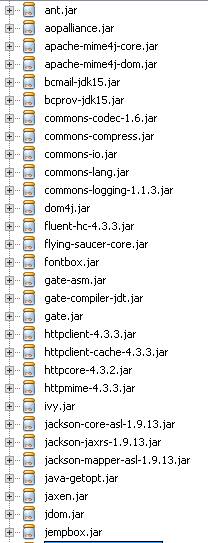
| | -- webservice 🡪 classi java per la gestione dei webservice come Google Maps

|-- org 🡪 contiene le classi java per la gestione delle chiamate/risposte con JSON (<http://www.json.org/java/>)

lib 🡪 le librerie jar utilizzate nel progetto

**Fig.33 : La struttura del nostro progetto java.**

I pacchetti *lib* e *org* contengono le librerie utilizzate nel progetto (*fig.34*), mentre il pacchetto *home* a tre sotto-pacchetti dediti rispettivamente alla estrazione dei ripsettivi oggetti java contenuti nel sotto-pacchetto *objects* GeoDocument,InfoDocument e GeoDomainDocument. Gli elementi presenti nel sotto-pacchetto *utils* invece verranno descritti nei prossimi paragrafi del capitolo.



**Fig.34: A sinistra le librerie JAR utilizzate e a destra uno screenshoot del nostro progetto java in Netbeans.**

*Le tre diverse classi main java nel progetto permettono di ottenere tre diversi tipi di documenti.*

. *Tra le librerie usate ve ne sono altre, oltre a quelle di default di gate.*

1. **Gli oggetti Java del nostro progetto**

Per il nostro progetto abbiamo già accennato a tre diverse strutture dell’informazione che andiamo a estarre dalle pagine web , oguna di queste strutture è costruita attraverso degli oggetti java (semplici construttori con setter and getter per ogni parametro) che andranno utilizzati per inserire nelle apposite tabelle GeoDocument e Info Document . Andiamo a vedere velcoamente la loro struttura e funzionalità che si rispecchia nella struttura delle ripsettive tabelle nel databaseMySQL dove l’unico elemento aggiunto è un identificatore (doc\_id) per una migliore gestione delle tabelle:

* **I GeoDocument,** il più semplice dei documenti è costiuito da un id, l’url della pagina web di riferimento , il nome dell acitttà e le coordinate geografiche, è la struttura base più semplice integrata nel processo solo per un eventuale integrazione diretta con le tabelle del progetto OntoStar.

GeoDocument : (url,city,latitude,longitude)

* **Gli InfoDocument,** uguali ai GeoDocument (utilizzano lo stesso oggetto java GeoDocument.java) costituiscono il cuore di questo progetto mostrando per un dato indirizzo web (url) tutte le informazioni del caso.

InfoDocument : (url,regione,provinci,city,indirizzo,iva,email,telefono,edificio,latitude,longitude,nazione)

* **I GeoDomainDocument,** sono il risultato finale di questo progetto costituiscono i GeoDocument o InfoDocument (a se conda di quale struttura si scelga) per i rispettivi domini web di tot. InfoDocument relativi per quel domini, essi sono il risultato di un’analisi statitisca dei GeoDocument/InfoDocument che sarà descritta nel dettaglio nel paragrafo 15 e nella validazione del paragrafo 16.

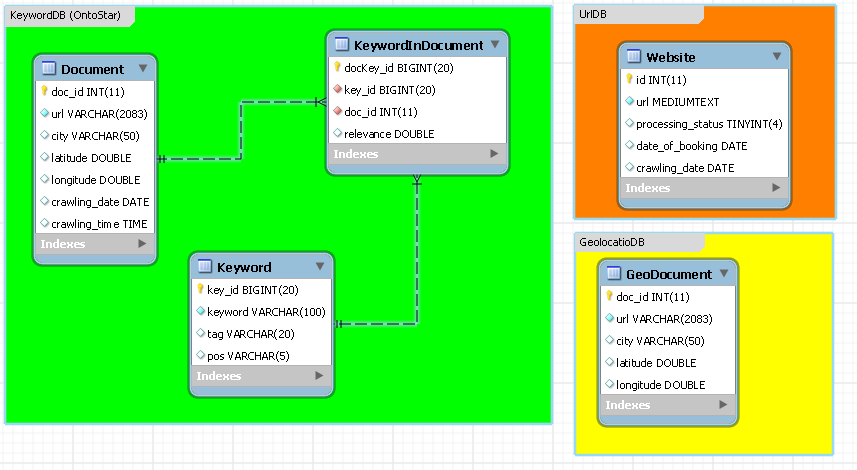
GeoDomainDocument: (url,regione,provinci,city,indirizzo,iva,email,telefono,edificio,latitude,longitude,nazione)

* **AnnotationInfo,** sono documenti temporanei alll’interno del progetto java utlizzatti per immagazzinare i contenuti estratti dalle varie regole jape con GATE, permettono di gestire meglio la “ripulitura”
* **FrequencyInfo e DepositFrequencyInfo,** sono documenti temporanei utilizzati nell’analisi statitisca per la realizzazione degli dei GeoDomainDocument i primi sono utilizzati per memorizzare i GeoDocument fina al limite imposto dal programmatore e poi su die ssi svolgere un’analisi statistica, i secondi la stessa cosa ma per gli InfoDocument.

1. **I database del progetto**

Come si vede dalla *fig.35*, nel nostro progetto avremo a che fare con tre database *KeywordDB* del sistema OntoStar, *UrlDB* e il nostro *GeolocationDB* che sarà creato per il progetto:

* UrlDB, contiene una tabella Website che non è altro che una lista di 6.000.000 di url.
* GeolocationDB, è un database contenente le tabelle GeoDocument e InfoDocument che contengono un record relativo ad un distinto url,ognuno dei quali è riferito a una pagina web che fa riferimento a un’url della tabella Website del database UrlDB.
* KeywordDB, contiene l’insieme di database utilizzati dal sistema OntoStar, in particolare ci interessa la tabella Document per integrare in modo più completo il risultato del nostro progetto.



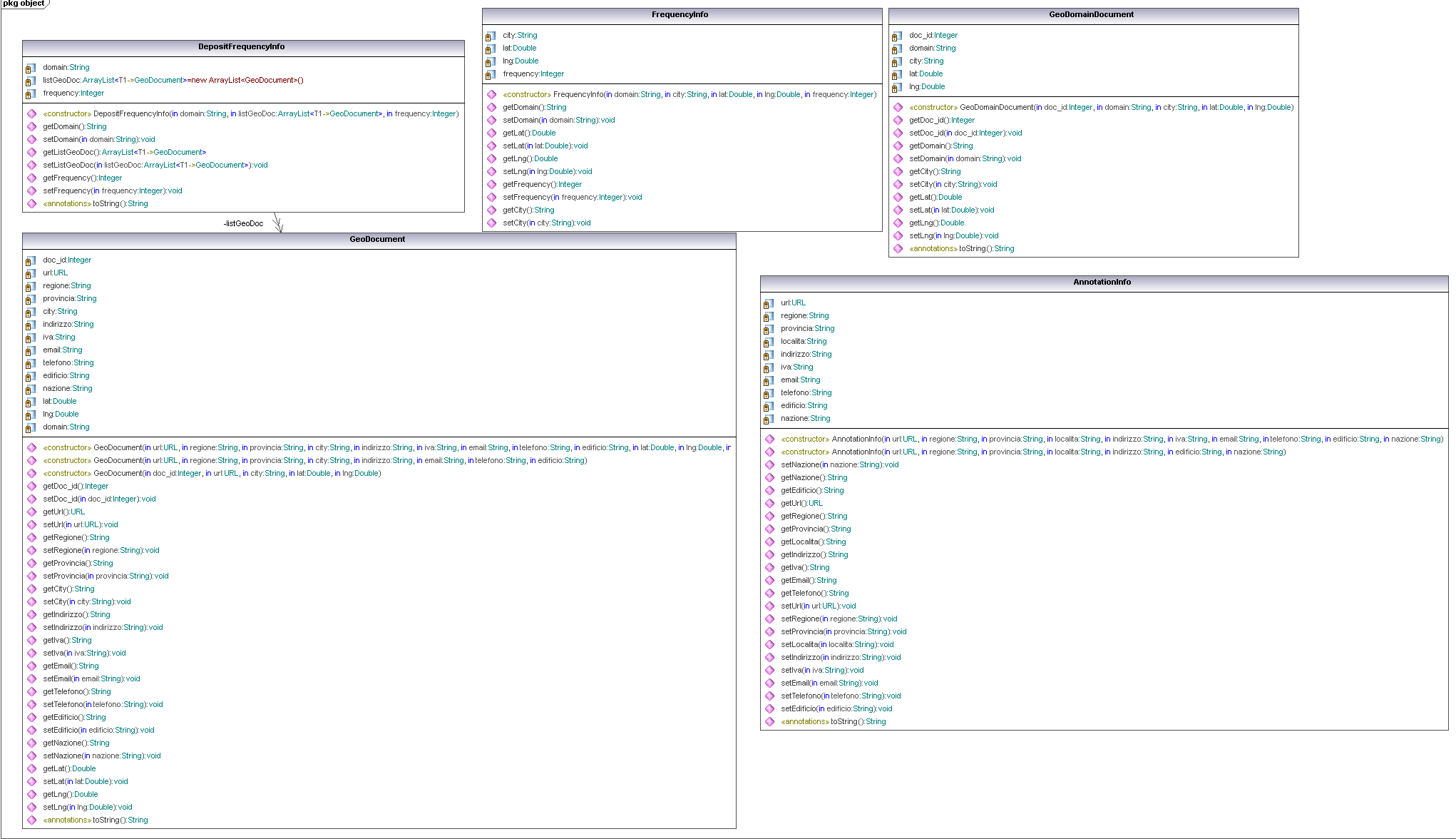
**Fig.35: I database utilizzati all’inizio del nostro studio visualizzati con MySql Workbench.**

La prima cosa da fare con il nostro programma è collegarci al database UrlDB e prelevare tutti gli url contenuti nella tabella website, quindi andiamo ad aggiungere alla nostra classe main la classe *ConnectToDatabaseMySql.java* che gestisce tali operazioni con i metodi *openConnectionTodatabase, closeConnectionToDatabase e getInfobyUrlDB* (*fig.36b*).

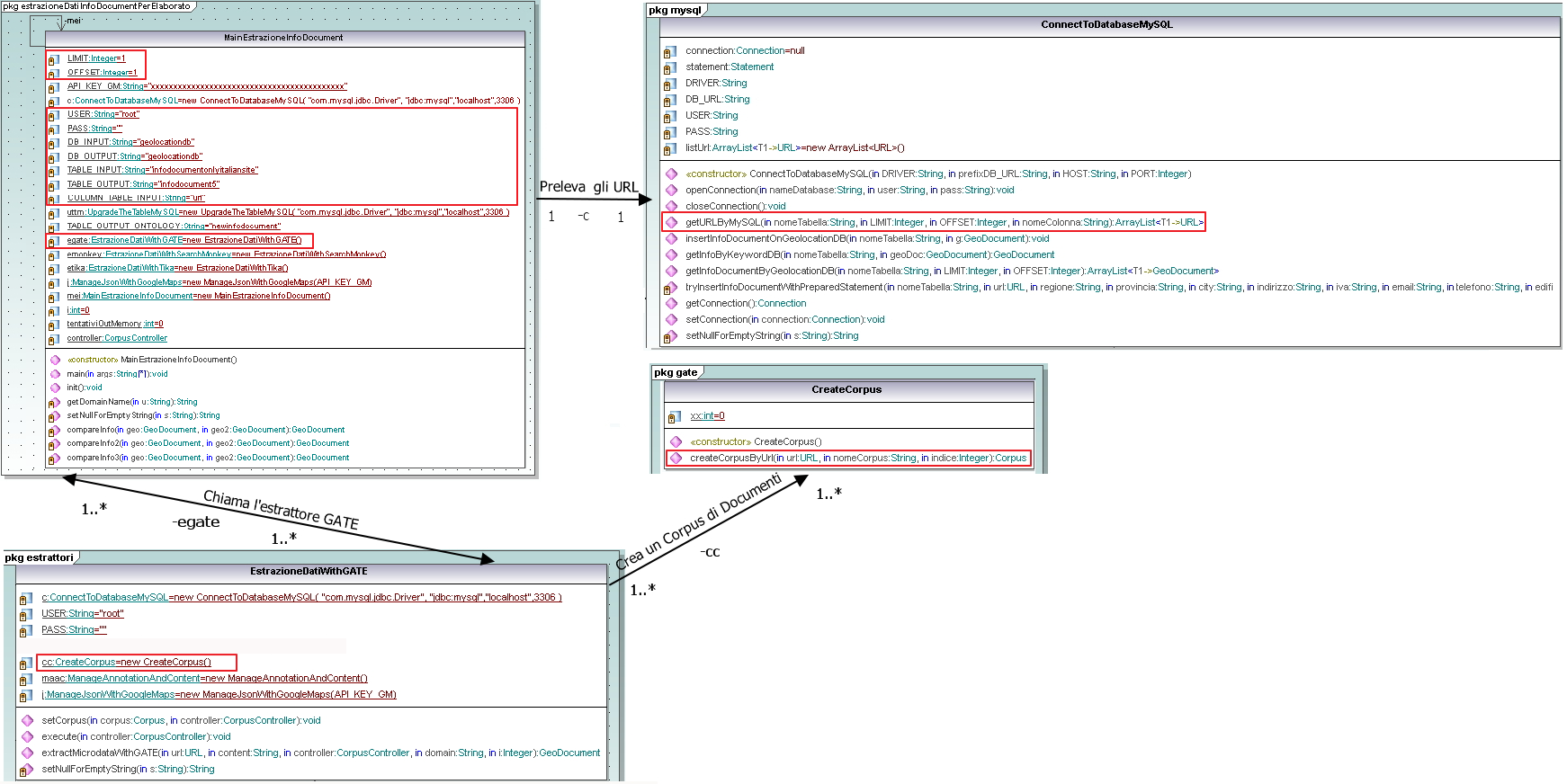
1. **Costruzione del Corpus degli URL**

Data una lista di URL, presi dal database UrlDB fornito, tentiamo di costruire per ognuno di loro un documento (Document) di GATE, attraverso un’opportuna classe JAVA che utilizza CORPORA per convertire il contenuto delle pagine web in un documento taggato in cui è possibile gestire le varie annotazioni e feature con GATE. Ogni volta che un documento GATE è costruito con successo, esso è aggiunto al Corpus, che alla fine della lista di url, sarà composto di tutti quei documenti che fanno riferimento a url ancora operativi o che non hanno un firewall che ne impedisce la visione senza autenticazione. Vediamo come varia il progetto aggiungendo la classe *CreateCorpus.java*.

Nei diagrammi UML che andremo a mostrare via via con la costruzione del nostro progetto, è stato deciso per una migliore “leggibilità” del progetto di trascurare le relazioni e i collegamenti con i diagrammi delle classi inerenti agli oggetti Java GeoDocument,AnnotationInfo,ecc.



**Fig.36: Diagramma UML del punto 7.5, gli oggetti java utilizzati nel progetto**



**Fig.36: Diagramma UML punto 7.6 e 7.7, connessione con il database MySQL e estrazione della lista di url da utilizzare e creazione del Corpus di GATE.**

1. **Configurazione locale di GATE**

Abbiamo inizializzato GATE, abbiamo implementato in GATE il nostro file .gapp/.xgapp e abbiamo creato un Corpus di documenti dagli URL forniti adesso vogliamo rendere il nostro progetto portatile su ogni computer e sistema operativo o in altre parole allocarlo all’interno del progetto JAVA.

Nel file .gapp/.xgapp, notiamo che tutti i relativi riferimenti a liste, grammatiche, plugin e regole Jape, si basano sulla GATE\_HOME di default di GATE cioè, dove è stato installato il software nel caso di Windows.

C:\Program Files\GATE\_Developer\_7.1\ e i vari elementi della pipeline cui fanno riferimento rispetto alle loro risorse su tale valore della path:

…………………………………………………………………………………………….

<!—Invochiamo il plugin di ANNIE nel file .xgapp🡪

<gate.util.persistence.PersistenceManager-URLHolder>

<urlString> $relpath$../../ANNIE/</urlString>

</gate.util.persistence.PersistenceManager-URLHolder>

Che sta dicendo che ANNIE è nella seguente cartella C:\Program Files\GATE\_Developer\_7.1\plugins\ANNIE **noi abbiamo lavorato sul nostro programma per renderlo disponibile su qualsiasi altro sistema operativo e computer, su GATE Cloud per altri utenti GATE o nel caso di un aggiornamento di GATE (com’è successo da 6.1 a 7.1) che trasforma completamente dei plugin che diventano inutilizzabili per il mio progetto.**

E’ necessario configurare localmente GATE all’interno del programma Java in modo tale che tutte le risorse faranno riferimento a GATE dalla directory del progetto Java. Un altro scopo del nostro studio è, infatti, quello di integrare un algoritmo di ricerca semantica che si “appoggia” a GATE, per il sistema OntoStar, per far questo è necessario rendere “portable” (portatile) il nostro progetto. Per far questo è necessario stabilire una nuova GATE\_HOME con tutte le relative path alle risorse questo, può essere fatto con pochissime righe di codice:

1. Creare nel nostro programma JAVA una cartella che chiameremo gate\_files che renderemo la nostra nuova GATE\_HOME riguardante il nostro programma: ***Gate.setGateHome(new File("gate\_files"));***
2. Copiare dalla cartella plugins di GATE i plugin che utilizziamo all’interno della nostra nuova GATE\_HOME la cartella *gate\_files*, chiamiamola *plugins*: ***Gate.setPluginsHome(new File("gate\_files/plugins"));***
3. Creare un nuovo file *gate.xml* per la configurazione del sito all’interno di *gate\_files* salvo particolari utilizzi si utilizza quello di default di GATE con copia-incolla: ***Gate.setSiteConfigFile(new File("gate\_files/gate.xml"));***
4. Creare un nuovo file *user-gate.xml* per la configurazione del sito all’interno di *gate\_files*

***Gate.setUserConfigFile(new File("gate\_files/user-gate.xml"));***

1. Creare all’interno di *gate\_files* una cartella *custom* contenente le cartelle *gapp, gazetteer, japes* contenenti a loro volta i file *.gapp/.xgapp, /.def .jape* usati. Esse devono essere all’interno della cartella dichiarata GATE\_HOME se vogliamo evitare di utilizzare riferimenti assoluti.

La struttura del nostra GATE\_HOME personale (la cartella *gate\_files*) è molto semplice:

gate\_file -- | -- custom---| --- gapp

| | --- gazetteer

| | --- japes

| | --- owl\_ontology

|-- plugin

**Fig. 37: Struttura della cartella gate\_files il nostro presonale GATE embeddedato nel progetto.**

Riconfiguriamo la classe main di estrazioneDatiInfoDocumentPerElaborato:

public static void main(String args[]) throws InterruptedException, InvocationTargetException{

EventQueue.invokeLater(new Runnable() {

public void run() {

// initialise the GATE

try{

Out.prln( "Initialising GATE..." );

//ATTENZIONE: Ora abbiamo reso portatile GATE all’interno del programma JAVA aggiungendo quattro linne di codice

Gate.setGateHome(new File("gate\_files"));

Gate.setPluginsHome(new File("gate\_files/plugins"));

Gate.setSiteConfigFile(new File("gate\_files/gate.xml"));

Gate.setUserConfigFile(new File("gate\_files/user-gate.xml"));

Gate.init();

Out.prln( "...GATE initialised" );

…………………………………………………………………………………..

} catch (Exception e1) {

Err.prln( "ECCEZIONE DI QUALCHE TIPO CAUSATA DA GATE" );

e1.printStackTrace();

}

} //run

}); //runnable

} //main

}

**Fig.38: Come varia la classe main() di estrazioneDatiInfoDocumentPerElaborato.java allocando GATE EMBEDDED all’interno del progetto JAVA**

Ora tutte le risorse del file .gapp/.xgapp utilizzeranno il riferimento alla GATE\_HOME locale all’interno del progetto JAVA con la variabile $gatehome$/path/to/file:

…………………………………………………………………………………………….

<!—Invochiamo il plugin di ANNIE nel file .xgapp🡪

<gate.util.persistence.PersistenceManager-URLHolder>

<urlString> $gatehome$custom/plugins/ANNIE/</urlString>

</gate.util.persistence.PersistenceManager-URLHolder>

1. **Estrazione e gestione del contenuto delle annotazioni**

Adesso dobbiamo prendere il contenuto del testo annotato dalle nostre annotazioni di GATE sui documenti del Corpus creato e gestire il loro “filtraggio” a livello Java, in modo da pulire al meglio quello che le nostre regole JAPE hanno “catturato” come informazione utile. Descriviamo alcuni dei concetti utilizzati.

In questo caso abbiamo il problema che le nostre regole Jape prendono tutti i possibili risultati che verificano il suo pattern(LHS), per esempio:

*Università degli Studi di Firenze Polo Scientifico e Tecnologico*

*Unifi – Università degli Studi di Firenze Polo scientifico e Tecnologico di Sesto*

*Università degli Studi Polo Scientifico e Tecnologico di Sesto Fiorentino*

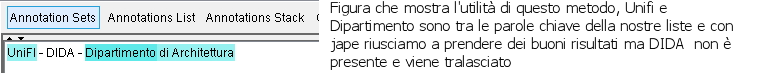
Nel primo caso la soluzione ideale è la seconda, nel secondo caso la terza, quindi qui diamo delle ipotesi su quale possa essere il risultato migliore fornito dalle regole Jape a seconda se si stia analizzando un documento HTML, o no,vediamo come il risultato migliore si ottiene a secondo di dove viene trovata l’informazione nel documento, nel nostro caso per esigenze di velocità, forniremo un’unica ma potente distinzione per la gestione dell’informazione annotata nel documento.

I meccanismi utlizzatti per “velocizzare” l’analisi dei documenti HTML sono applicabili a qualunque documento taggato che GATE riesce a gestire e.g. XML,PDF addirittura esiste un plugin per i file Json.

**Nel caso il documento HTML abbia l’informazione nell’HEAD o il FOOTER:**

* Come succede spesso molte delle informazioni sulla pagina web si trovano o sul Fotter o Header del documento html, all’interno di essi le regole Jape trovano diverse annotazioni accettabili ma di solito il contenuto più lungo che ha rispettato il pattern è anche quello più specifico e quindi migliore.
* Verificare che date due parole chiave si faccia un “ponte” del contenuto, per esempio:

*“Unifi – Università degli Studi di Firenze bla bla bla di chimica”,*Tutte le informazioni non considerate buone dalla regola Jape posso essere invece utili all’utente semplicemente perché di solito, è molto probabile che vi siano altri ingredienti del panino tra le due frasi che possono essere utili. Questo ci permette di non riempire le nostre con ogni singola parola chiave possibile. Questo procedimento è attuabile solo nell’HEAD e FOOTER del documento o parti del testo limitate di piccole dimensioni, se lo facessimo nel BODY due parole chiave molto distanti tra di loro prenderebbero un grosso pezzo di testo inutile.



Si potrebbe tranquillamente inserire DIDA nel nostro Gazetter per completare l’annotazione, ma avendo sei milioni di pagine da analizzare forse ci conviene questo metodo piuttosto che inserire milioni di voci nel gazetter almeno per quanto rigurada il nostro studio.

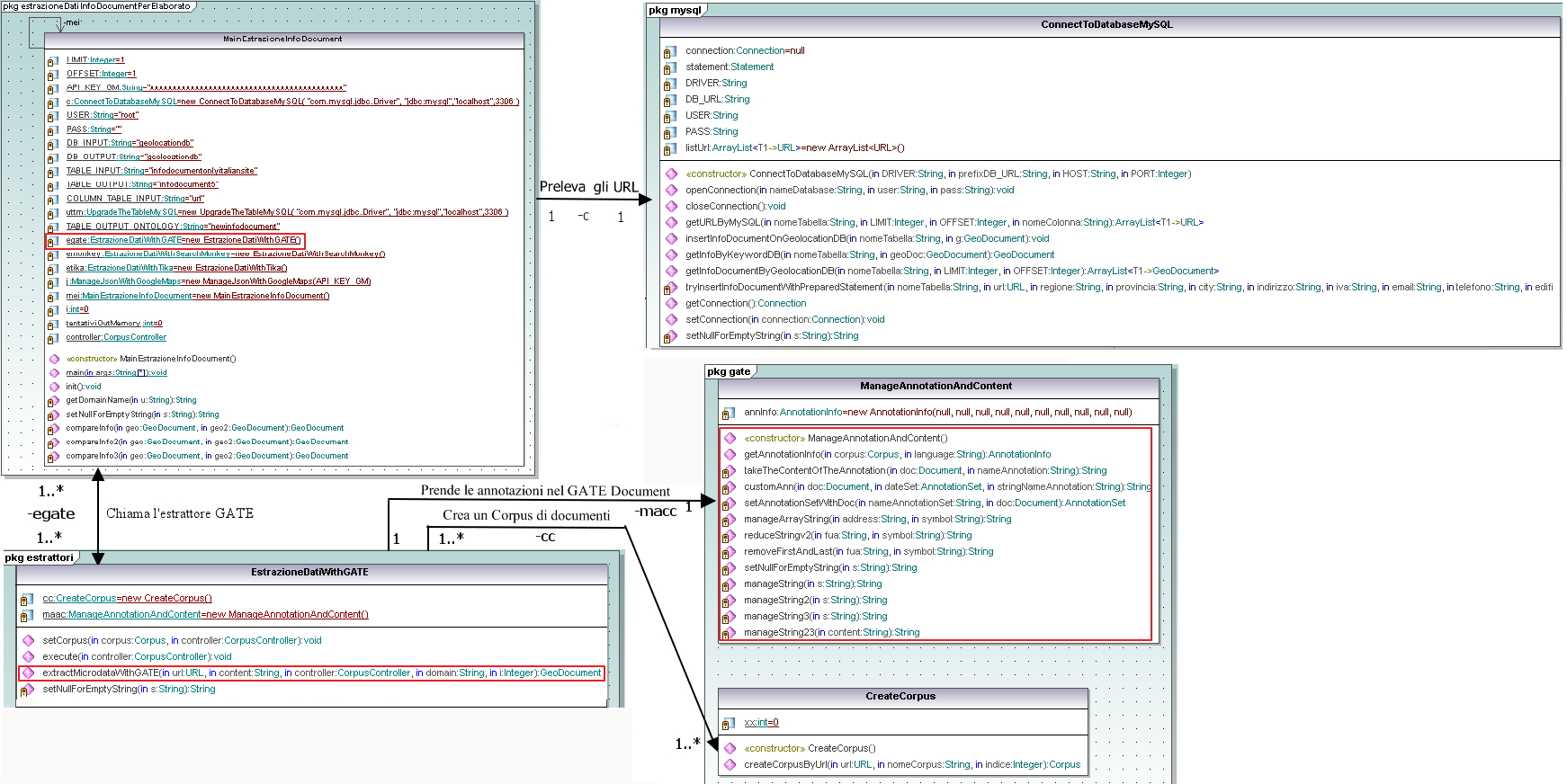
**Nel caso il documento HTML non abbia le informazioni nell’HEAD o FOOTER ma le abbia nelle particolari sezioni del testo adibite ad esse:**

* A volte analizzando le pagine web si nota che le informazioni che ci interessano non sono in una sezione del documento html taggata non con footer ma da altri elementi che fanno il ruolo di footer per esmpio un div con id “footer” che fanno la stessa identica cosa ma sono necessarie regole jape diverse.
* Molto comuni sono i div in fondo al documento con id del tipo “footercc” oppure “footercredits”, per tenere conto di queste varianti è necessario imporre le regole JAPE idFooter,idHeader e idSpecial. per queste porzioni di testo e analizzarle come footer e header di “riserva”. Insomma l’ultima spiaggia prima di andare ad analizzare l’enorme body del documento.

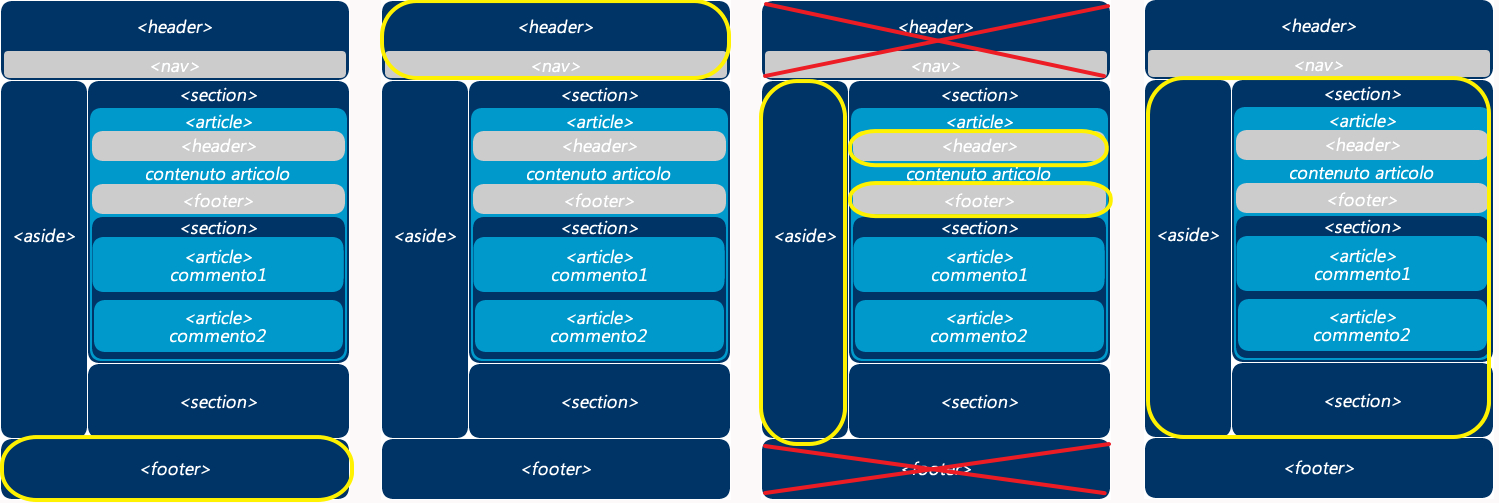
**Nel caso il documento non sia HTML o si sia costretti ad analizzare il body del documento:**

* Andiamo a prendere ogni singolo contenuto di ogni singola annotazione questo provoca molte informazioni ridondanti per esempio “Università degli Studi” e “Università degli Studi di Firenze” rispettano entrambi il pattern Jape e sono segnate come annotazioni differenti, per evitare di prendere due o più volte la stessa informazione è stato necessario introdurre metodi di analisi e gestione delle stringhe in java che rallentano il programma, ma forniscono un risultato molto più pulito e di qualità.
* Sono stati previsti diversi metodi di gestione delle stringhe Java per filtrare al meglio un risultato del contenuto, in modo che fosse il più breve e completo possibile, non ne verranno descritti i dettagli perché si tratta di meccanismi di programmazzione e non di “tecniche” per un corretto tracciamento dell’informazione.

Tutto il meccanismo in precedenza descritto nei vari casi è svolto dalla classe java *ManageAnnotationAndContent.java* in *figura 39,* mentre in *figura 40*, vi è una rappresentazione grafica di come lavora il meccanismo di analisi di questa classe in un documento hmtl.



**Fig.39: Diagramma UML punto 7.8, Aggiunta della classe ManageAnnotationAndContent.java per l’estrazione del content.**



**Fig.40: Rappresentazione grafica delle varie fasi di analisi del documento HTML.**

1. La prima fase: analizza le informazioni all’interno del footer del documento HTML se presente.
2. La seconda fase: se non tutte le informazioni sono state trovate nel footer allora analizza le informazioni all’interno dell l’header del documento HTML se presente.
3. L a terza è il caso particolare dove la pagina non ha né il footer né l’header , ma ha comunque degli opportuni sostituti all’interno dei tag HTML con particolari valori di id o class,(e.g. <div id=”footer”>).
4. La quarta è ultima fase è l’analisi di tutto il corpo del documento, o meglio di tutto ciò che non è stato analizzato precedentemente, è inutile infatti stare ad analizzare di nuovo il footer e l’header delle fasi precedenti, che non hanno fornito per niente o in parte le informazioni che volevamo, è naturale che quelle poche informazioni che avevano sono state estratte con successo e salvate precedentemente per essere integrate con quelle che saranno estratte dal body del documento se presenti.
5. Si potrebbe dire che vi è una quinta fase nascosta automatica di GATE per l’analisi dei documenti, ma essa riguarda altri formato come il PDF, dove non vi è una esplicita tagenizzazione del documento.

**Ricordiamo che il processo di analisi viene fermato non appena sono trovate tutte le informazioni di cui abbiamo bisogno, per esempio se troviamo tutto nel footer del documento le fasi 2,3,4 non verranno mai lanciate il processo sarà interroto e passeremo al documento successivo.**

1. **Configurazione chiamata/risposta con l’API Google Maps**

Abbiamo una lista di oggetti Java *AnnotationInfo* ognuno riguardante un documento web analizzato, annotato ed elaborato da GATE con cui sono state estratte le informazioni individuate dalle nostre regole JAPE. Adesso è necessario ricavare da tali informazioni le coordinate geografiche latitudine e longitudine per creare, per ogni *AnnotationInfo,* un oggetto *GeoDocument* che costituiscono i record della tabella *GeoDocument* che abbiamo creato nel nostro database GeolocationDB.

I nostri *GeoDocument,* per essere interlacciati con il sistema OntoStar e in particolare con la tabella *Document,* hanno bisogno di due campi, Latitude (latitudine) e Longitude (longitudine) avendo nelle nostre *AnnotationInfo* valori come indirizzo, località, ecc. Una soluzione molto semplice è quella di utilizzare l’API Google Maps, dove dobbiamo dalle nostre AnnotationInfo precedentemente create ricavarci la latitudine e longitudine del luogo/località a cui fa riferimento il contenuto del documento web.

E’ necessario quindi invocare da codice JAVA il servizio Web di Google Maps, prendere una chiave per il development[[36]](#footnote-36) e abilitare l’utilizzo del Geocoding[[37]](#footnote-37). Una volta fatte tutte queste azioni lato “server” sono necessarie delle scelte progettuali da parte nostra:

1. Le chiamate all’API Google Maps sono realizzate con il GET http.
2. Le risposte dall’API Google Maps sono nel formato JSON.

Avendo scelto JSON come formato di risposta dell’API Google Maps alle nostre richieste è necessario creare delle corrette classi a livello JAVA per gestire il tutto. Il problema non sussiste, tali classi JAVA sono già fornite su vari siti di diversi autori. Noi ci limiteremo a importarli nel nostro progetto, in particolare abbiamo utilizzato la versione ufficiale di JSON per l’integrazione in JAVA[[38]](#footnote-38).

La prima cosa che dobbiamo fare con il nostro codice è concatenare tutte le informazioni utili delle *AnnotationInfo* e comporle in un formato url da inviare all’API Google, per esempio:

***Università degli Studi di Firenze, \_Toscana*** con lingua “**it**” ricavata dall’ identificazione della lingua

deve diventare

***address=Università+degli+studi+di+Firenze+Toscana&region=it***

Questo è realizzato, con un paio di metodi di formattazione delle stringhe, tale stringa “ritoccatta” sarà inserita opportunamente secondo la struttura del Geocoding nella richiesta URL per l’API Google Maps. Il tutto sarà poi gestito dalla classe *ManageJsonWithGoogleMaps.java* in *fig.42.*

*Il vantaggio tanto acclamato in precedenza sul bisogno o meno di identificare la lingua del documento sta nel parametro region di Google Maps semplicemente perché se scriviamo Firenze&region=it , sa per certo che deve cercare la città di Firenze in Italia e non la città di Firenze in Asia se esiste. Il guadagno di tempo in questo senso ci permette di spendere tot. Millisecondi nel tracciamento della lingua.*

*La licensa free di Google Maps permette solo 3000 query al giorno per “sovverchiare” il problema utilizziamo un metodo java ribatezzato Temporizzatore delle query che metto disponibile in APPENDICE*

Le nostre regole Jape non sono perfette o meglio non lo sono le pagine web analizzate, che non hanno a volte nessuna informazione di geo localizzazione, quindi non tutte le AnnotationInfo avranno sufficienti informazioni per Google Maps da potergli far restituire un risultato. In questo caso per i GeoDocument corrispondenti scegliamo di lasciare a null i valori dei campi Latitude e Longitude.

1. **Integrazione dei dati con OntoStar e inserimento dei GeoDocument nel database GeolocationDB.**

Il nostro progetto è quasi finito, applichiamo la nostra ultima scelta progettuale per cui se il valore del campo city del nostro *GeoDocument* è null, lo sostituiamo con il valore del campo city del corrispondente record della tabella *Document* del database KeywordDB del sistema OntoStar che fa riferimento allo stesso url cioè alla stessa pagina web. Si aggiunge il metodo *getInfoByKeywordDB*, alla classe *ConnectToDatabaseMySql.java* per svolgere tale operazione. Alla fine del nostro programma JAVA abbiamo realizzato una lista di *GeoDocument* che inseriremo nel database *GeolocationDB,* anche questo aggiungendo un semplice metodo *insertGeoDocumentOnGeolocationDB*, alla classe *ConnectToDatabaseMySql.java*. in *fig.43* è mostrato il diagramma UML.cone videnziato il metodo.

1. **Integrazione dei dati con il motore di ricerca Yahoo SearchMonkey**

Come già accennato siamo ben lontani da tutti quei meccanisimo di ricerca semantica che i motori di ricerca applicano sul web per la ricerca dei risultati ma questo non ci vieta di sfruttarli, anche se in modo molto brusco.

Il meccanismo è motlo semplice, forniamo al motore di ricerca Yahoo SearchMonkey attraverso un comando POST l’url della pagina da analizzare, yahoo restituira la lista dei suoi risultati, attraverso l’utilizzo della classe estrazioneDatiWithSearchMonkey.java, il nostro programma “prende” il primo risultato che è quasi sicuramente quello giusto e analizza il dom della pagina html restituita da yahoo.

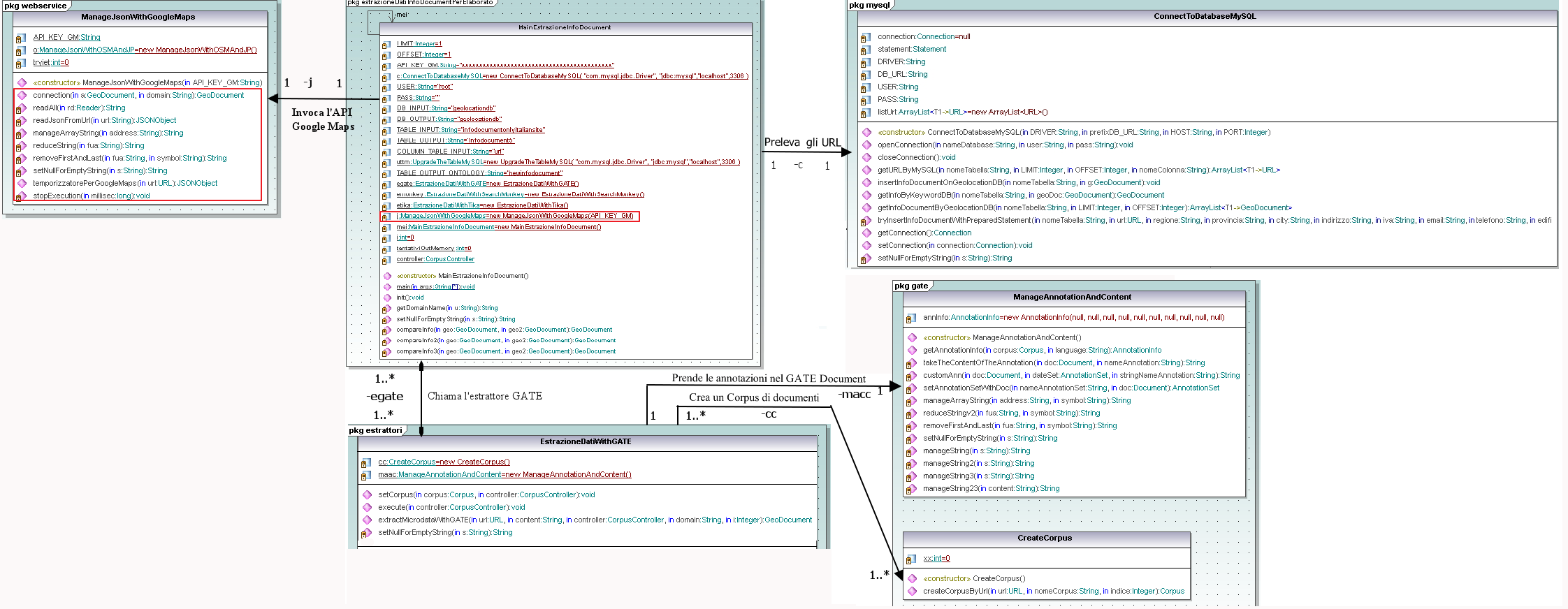
Come si è notato in tutti i motori di ricerca i risultati della varie pagine ottenute dalla ricerca hanno cone se delle informazioni e.g. digitate www.unifi.it su google sotto il link di rifeirmento alla pagina dell’università vi sono informazioni come indirizzo,telefono,ecc.



**Fig.41 : Screenshoot che mostra l’utilità di questo metodo.**

**Conviene farlo?**

* Se tutto va come previsto il risparmio di tempo tra analizzare con GATE un’intero documento HTML e le “due righe” in evidenza in *fig.41* è enorme e per l’analisi di sei milioni di documenti la velocità fa molto comdo. In *fig.43*, si uò notare come la classe estrazioneDatiWithSearchMonkey.java richiama la classe estrazioneDatiWithGATE.java fornendo come testo del GATE document da analizzare il testo estratto dal dom della lista risultati di SearchMonkey/Yahoo.
* Vi è inoltre un meccanismo di controllo aggiuntivo che prevede se nel caso l’indirizzo (l’informazione più importante) o altre informazioni cruciali non siano presenti nelle “due righe” si deve comunque chiamare GATE sul documento, in questo sfortunato caso abbiamo *rallentato* il programma invece di *velocizzarlo.*
* Mostreremo nel capitolo 8 una tabella relativa su un dataset di documenti e se statisticamente utilizzare questo metodo conviene o meno.



**Fig.42: Diagramma UML punto 7.9, integrazione di Google Maps nel progetto.**



**Fig.43: Diagramma UML punto 7.11 e 7.12, aggiunta estrazione con SearchMonkey.**

1. **Integrazione dei dati con Apache Tika**

Apche Tika è uno strumento scritto in Java per l’identificazione e l’estrazione di metadati e testo da numerosi tipi di documenti. Si tratta quindi di un software molto utile per il recupero di dati sia da fonti esterne (Internet),sia da fonti interne, cioè il patrimonio aziendale di documenti. I formati da cui può estrarre i dati Tika sono innumerevoli HTML,XML,Microsoft Office, PDF, ecc. Tika esegue non soltanto l’estrazione di metadati e testo, ma anche il riconoscimento della lingua in cui il documento è scritto.

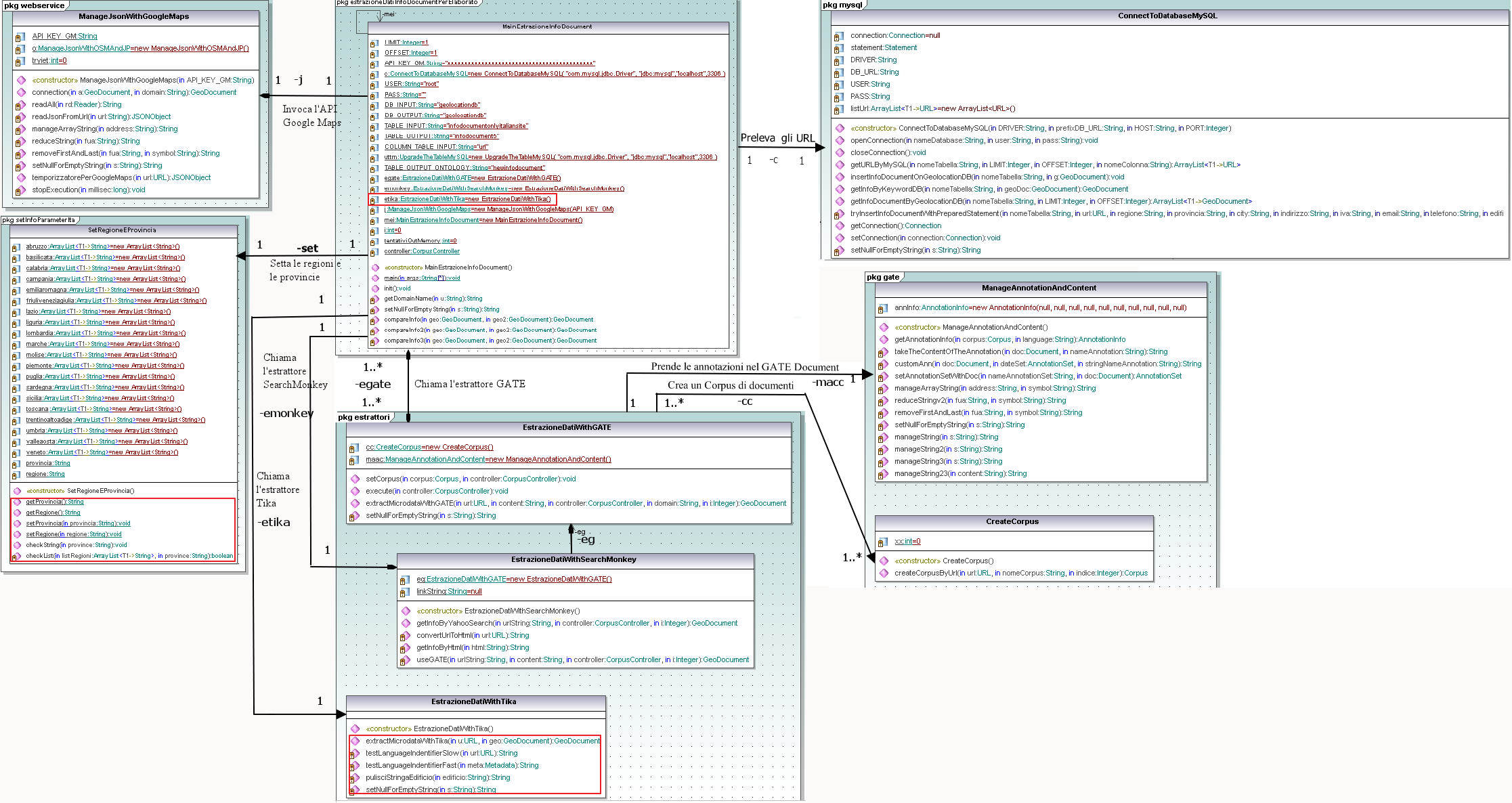
In fig.34 è possibile notare che GATE fornisce già le librerie di Tika e quindi in teoria non c’è bisogno di scaricare la libreria di Tika dal sito ufficiale, ma per una migliore elaborazione dei vari tipi di documento (per esempio le librerie “base” di Tika in GATE non tengono conto del formato MBOX il formato e-mail utilizzato da molti archivi), sarà utilizzata la versione ufficiale del sito.Con Tika vengono estrarre le informazioni inerenti a <title> <description> e <language>.

**Conviene farlo?**

* Le metainformazioni estratte con Tika sono già, estratte dalle nostre regole Jape e anche dal integrazione di SearchMonkey, l’utilità di Tika è semplicemente la velocità che abbiamo già accennato come una delle qualità che deve avere il nostro programma. Statisticamente parlando nel circa 70% dei casi di estrazioneSearchMonkey si estraggono solamente le informazioni inerenti alle regole MyEdificio,MyNazione che corrispondono per come le abbiamo pensate alle metainformazioni <title> e <language>. Mostreremo nel capitolo 8 una tabella che mostra i vantagi e svantaggi delle varie integrazioni in un dataset di documenti.
* Oltre alle metainformazioni Tika permette anche un’ottima analizzi dei documenti per l’identificazione del linguaggio, ricordiamo che abbiamo realizzato delle regole jape che utilizzano gli strumenti di GATE per l’analisi del linguaggio come Lingpipe,ma ci sono dei fattori di cui tenere conto:
  + tali strumenti si apppoggiano su Tika per le loro analisi quindi la si può utilizzare direttamente nel nostro codice java per identificare la lingua in cui è scritto documento.
  + Con le nostre regole Jape eravamo limitato a documenti taggati come XML e HTML mentre Tika a un numero di formati analizzabili molto più ampio.
  + I plugin di GATE che abbiamo utilizzato per l’analisi del linguaggio come LingPipe forniscono molte utilità, ma per l’identificazione del linguaggio i basano proprio su Tika.
* L’unico fattore negativo di Tika per il nostro studio è il tempo di elaborazione comunque molto corto.
* Mostreremo nel capitolo 8 una tabella relativa su un dataset di documenti e se statisticamente utilizzare questo metodo conviene o meno.

1. **Integrazione delle Regioni e delle provincie**

Nelle pagine Web aziendali e non, tra le informazioni utili alla geolocalizzazione dell’azienda vi è fornito di solito solo il nome della città di appartenenza. Per il nostro progetto abbiamo necessità in alcuni casi (per esempio per un miglior risultato da parte di Google Maps di ricavarci anche la regione e la provincia, per far questo si utilizzano delle regole jape MyRegione MyProvincia e MyLocalita accennate precedentemente nel paragrafo delle regole jape. Spesso però l’abbinazione Regione 🡪 Provincia🡪città risulta errata non per colpa dell’estrazione di GATE con le regole jape, ma perché vi sono troppe varabili nei documenti , la regione ricavata dalla regola MyRegione “dovrebbe” essere quella giusta, ma non è detto che essa corrisponda alla città di residenza dell’azienda che è stata estratta con MyLocalita, la classe java che andremo ad aggiungere al nostro progetto fa un controllo sulla relazione città 🡪 Provincia 🡪Regione e verifica se è a posto, se non lo è si occupa di settare dei nuovi valori nel campo Provincia e Regione per la relativa città. Il controllo è molto leggero e veloce e la riduzione di velocità del programma dovuto ad esso è irrilevante.



**Fig.44a: Diagramma UML del punto 7.13 e 7.14, aggiunta della classe setRegioneEProvincie.java e EStrazioneDatiWithTika.java**

1. **Ultima classe java: Upgrade della tabella MySQL per la successiva integrazione con un’Ontologia**

Benchè fuoriesca dall’ ambito del nostro studio, viene inserita un’ulteriore classe java UpgradeMySQLTable.java , essa richiama appositi comandi MySQL con opportuni meccanismi java per “preparare” i nostri InfoDocument a essere integrati con una particolare Ontologia per esempio creando appositi campi identificativi per la funzione di URI per ogni singola classe dell’Ontologia che andremo ad utilizzare.

Tale classe viene mostrata in figura 48 rappresentante il diagramma UML finale del progetto.

1. **Creazione della tabella GeoDomainDocument per la successiva validazione del progetto.**

Una volta finito di creare la nostra tabella di GeoDocument/Infodocument con record che fanno riferimento ad ogni singolo URL, è necessario realizzare una relazione fra i domini web di tali URL con le corrette informazioni geografiche, per semplicità d’uso nella spiegazione utilizziamo la struttura dei GeoDocument, nel nostro progetto è previsto comunque tale utilizzo anche per gli infodocument e quindi un’analisi non solo delle coordinate geografiche, ma anche di tutte le altre informazioni come indirizzo ,email,ecc.

Il problema sussite quando analizzando più pagine web nello stesso dominio web si hanno dati estratti differenti si per errrore da parte del programma o della pagina stessaInfatti ci si può ritrovare nel caso in cui nella tabella GeoDocument:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DOC\_ID** | **URL** | **CITY** | **LAT** | **LNG** |
| 111234 | http://www.unfi.it/disit | Firenze | xxxxxxx | xxxxxxxx |
| 111235 | ftp://www.unifi.it/home | Florence | xxxxxxxx | xxxxxxxx |
| 111236 | http://www.unifi.it/siena | Siena | xxxxxxxx | xxxxxxxxx |

**Fig.45: Record della tabella GeoDocument**

In questo caso nel medesimo dominio www.unifi.it ci si può ritrovare ad avere moltecipli coordinate geografiche che fanno riferimento ai singoli URL e non all’host. Per tenere conto di questo è necessario fornito come input la tabella *GeoDocument* stabilire delle *regole di “identificazione delle correte informazioni geografiche”* cioè qualcosa che ci conferma al cento per cento che le informazioni per quel dato dominio web sono vere.

Oltre a decidere su quante pagine web andiamo a realizzare i GeoDomainDocument dobbiamo anche imporre un valore che chiamiamo ***range*** cioè quanti record di GeoDocument/InfoDocument che fanno riferimento allo stesso dominio dobbiamo “immagazzinare” nel progetto prima di analizzarne il contenuto, la varianza che diamo al valore range ci permette di avere diversi risultati nell’analisi statistica successiva. Questo è molto utile per permetterci di capire quante tot. Pagine per un dato dominio dobbiamo analizzare prima avere un risultato sicuro.

Per esigenze di velocità nell’analisi delle pagine impostiamo delle regole.

**Prima regola**: Se l’URL che fa riferimento alla pagina principale o home del dominio fornisce dei risultati li prendiamo come giuste e tralasciamo tutti gli altri record presenti nella tabella Geodocument visti o non visti che appartengono allo stesso dominio web (e.g. www.unifi.it, www.unifi.it/, www.unifi.it/index.html, ecc.).

**Seconda regola**: Se non è stato possibile indentificare le coordinate geografiche con la prima regola durante la lettura della tabella GeoDocument, si stabilisce che se almeno tot. risorse url (il nostro range di documenti) che fanno riferimento allo stesso dominio web forniscono le stesse coordinate geografiche allora le consideriamo vere e tralasciamo tutti gli altri record presenti nella tabella Geodocument visti e non visti che appartengono allo stesso dominio web. Se successivamente viene letto un’URL che rispetta la prima regola anch’esso viene tralasciato.

I risultati dell’applicazione di queste due regole sulla tabella GeoDocument implementate in java, vengono inseriti in una nuova tabella del database GeolocationDB GeoDomainDocument dove si fa riferimento ai singoli domini web e non ai singoli URL , mostriamo il risultato della tabella GeoDomainDocument:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DOC\_ID** | **DOMAIN** | **CITY** | **LAT** | **LNG** |
| 111234 | www.unfi.it | Firenze | xxxxxxxxx | xxxxxxxxxxxxx |
| 111235 | www.apretoscana.it | Toscana | xxxxxxxxxxx | xxxxxxxxxxx |
| 111236 | www.unisi.it | Siena | xxxxxxxxxxx | xxxxxxxxxx |

**Fig.46: Record della tabella GeoDomainDocument**

Adesso possiamo realizzare una validazione dei nostri risultati rispetto ai domini web che era esattamente cosa volevamo realizzare, non ci resta che testare il nostro progetto e vedere come si comporta.

Quello che è stato mostrato per i GeoDocument vale anche per gli InfoDocument si setta un limite di url da analizzare un range di documenti per singolo dominio, una volta che sono stati “immagazzinati” i FrequencyDepositInfo per singolo dominio web che rispettano il limite range, andiamo ad analizzare per ogni singolo campo di ogni singolo InfoDocument del DepositFrequencyInfo quale è il valore più diffuso tra le pagine web cioè quale è quello più frequente e quindi con più probabilità di essere il valore dell’informazione esatta che vogliamo estarre , in java ci sono centinaia di modi per far questo, addirittura si poteva fare attraverso query MySQL.

1. **Output del programma Java e Diagramma UML finale**

Il nostro progetto è finalmente concluso. Diamo un’occhiata al risultato della console Java per l’elaborazione di un documento web, il valore null del doc\_id finale viene cambiato in fase di inserimento del record grazie alla proprietà mySql di AUTOINCREMENT:

**file:///C:\Users\Marco\Desktop\DS ELABORATO\datastore\_001**

**Connection to the database....com.mysql.jdbc.JDBC4Connection@66857f69**

**lette informazioni dal database...**

**connessione con il database chiusa**

**Using gate\_files as GATE home**

**Using gate\_files\plugins as installed plug-ins directory.**

**Using gate\_files\gate.xml as site configuration file.**

**Using gate\_files\user-gate.xml as user configuration file**

**Using C:\Users\Marco\gate.session as user session file**

**CREOLE plugin loaded: file:/C:/Users/Marco/Desktop/%5BNetbeans%20Project%5DEstrazioneDatiWebPageToRecordSQL%2015-10-2014/gate\_files/plugins/Tools/**

**CREOLE plugin loaded: file:/C:/Users/Marco/Desktop/%5BNetbeans%20Project%5DEstrazioneDatiWebPageToRecordSQL%2015-10-2014/gate\_files/plugins/LingPipe/**

**CREOLE plugin loaded: file:/C:/Users/Marco/Desktop/%5BNetbeans%20Project%5DEstrazioneDatiWebPageToRecordSQL%2015-10-2014/gate\_files/plugins/ANNIE/**

**CREOLE plugin loaded: file:/C:/Users/Marco/Desktop/%5BNetbeans%20Project%5DEstrazioneDatiWebPageToRecordSQL%2015-10-2014/gate\_files/plugins/JAPE\_Plus/**

**CREOLE plugin loaded: file:/C:/Users/Marco/Desktop/%5BNetbeans%20Project%5DEstrazioneDatiWebPageToRecordSQL%2015-10-2014/gate\_files/plugins/Tagger\_Framework/**

**...GATE initialised**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Run GATE**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Documento http://www.ceprof.unibo.it non più disponibile o raggiungibile.**

**(0)Creating doc for http://www.cesal.unifi.it**

**(1)Creating doc for http://www.cesdop.it**

**……………………………………………………………………..**

**(45)Creating doc for http://www.dcp.unifi.it**

**Contenuto del Corpus costituito da:46 indirizzi url.**

**file:///C:\Users\Marco\Desktop\DS ELABORATO\datastore\_001**

**Datastore datastore\_001 aperto!!!**

**file:///C:\Users\Marco\Desktop\DS ELABORATO\datastore\_001**

**datastore\_001**

**Il datastore esiste già....**

**corpus saved in datastore...**

**[0-50](50)\_GeoDocuments\_Corpus\_\_\_1413791028462\_\_\_4144**

**GATE:GeoDocument{doc\_id=null, url=http://www.cesal.unifi.it, regione=null, provincia=Firenze , city=Firenze, indirizzo= Viale Pieraccini, 6**

**50139, iva=null, email=cesal@unifi.it**

**, telefono=+39.055.4271203, edificio= UniFI - Centro Stabulazione Animali da Laboratorio, lat=null, lng=null, nazione=null, description=null}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Run Tika**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**DESCRIPTION:null**

**EDIFICIO:UniFI - Centro Stabulazione Animali da Laboratorio**

**LANG:it**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**TIKA+GATE:GeoDocument{doc\_id=null, url=http://www.cesal.unifi.it, regione=null, provincia=Firenze , city=Firenze, indirizzo= Viale Pieraccini, 6**

**50139, iva=null, email=cesal@unifi.it**

**, telefono=+39.055.4271203, edificio=UniFI - Centro Stabulazione Animali da Laboratorio, lat=null, lng=null, nazione=it, description=null}**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**………………………………………………………………………………………………..**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*DOCUMENT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**CITY:Firenze**

**LANGUAGE:it**

**DOMAIN:www.cesal.unifi.it**

**[1]ALPHA 1:it DOMAIN:www.cesal.unifi.it**

**NAZIONE:Italia**

**URL for GM:https://maps.googleapis.com/maps/api/geocode/json?address=Toscana+Viale+Pieraccini+6+50139+Firenze&region=Italia&sensor=false**

**COORD[LAT:43.80264529999999,LNG:11.2453729]**

**………………………………………………………………………………………………………**

**Connection to the database....com.mysql.jdbc.JDBC4Connection@a37f250**

**INSERIMENTO**

**---------------------------------------------------------------------------------------------------------------------------------------**

**GeoDocument{doc\_id=null, url=http://www.cesal.unifi.it, regione=Toscana, provincia=Firenze, city=Firenze, indirizzo= Viale Pieraccini, 6 50139, iva=null, email=cesal@unifi.it**

**, telefono=+39.055.4271203, edificio=UniFI Centro Stabulazione Animali da Laboratorio, lat=43.80264529999999, lng=11.2453729, nazione=Italia, description=null}**

**---------------------------------------------------------------------------------------------------------------------------------------**

**Record is inserted into the table!**

**connessione con il database chiusa**

**………………………………………………………………………………………………………..**

**Fig.47: Output finale del programma java.**



**Fig.48: Diagramma UML finale.**

1. **Validazione del progetto e Conclusione**

Prima di parlare della validazione è bene soffermarci sull’utilizzo di quali estrattori delle informazioni ci conviene usare nel nostro studio. Nel corso della creazione del nostro progetto ne sono stati creati tre uno con GATE,uno con SearhMonkey e un altro con Tika. Andremo a dare un’occhiata a quale conbinazione delle varie metodologie di estrazione incontrate e analizzate si adatta di più ai nostri InfoDocument in termini di qualità e velocità. Per far questo saranno utilizzati due dataset prestabiliti, uno di circa 1000 url di pagine web qualunque e un dataset di circa 1000 url relativi a singoli domini web . Mille è un dataset molto piccolo ,ma sufficiente a vedere i vantaggi e svantaggi degli estrattori, qui di seguito mostriamo una tabella.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Dataset di url** | **GATE** | **SearchMonkey** | **Tika** | **Velocità Media per l’analisi di un documento** | **Qualità delle informazioni estratte** |
| Dataset (1000 url) | X | X | X | 25 secondi | 8/10 |
| Dataset (1000 url) | X | X |  | 20 secondi | 7/10 |
| Dataset (1000 url) | X |  | X | 15secondi | 7/10 |
| Dataset (1000 domain) | X | X | X | 25 secondi | 9/10 |
| Dataset (1000 domain) | X | X |  | 10 secondi | 10/10 |
| Dataset (1000 domain) | X |  | X | 15 secondi | 9/10 |

**Fig.49a: Tabella che mostra quale combinazione degli estrattori conviene usare per determinati dataset.**

Come si può vedere l’unica differenza sostanziale sta nel’utilizzo o meno di SearchMonkey per determinate pagine web, se sitratta o meno delle pagine principali di un sito ho di pagine di riferimento ad esse che forniscono al motore di ricerca di Yahoo le informazioni di geolocalizzazione volute.

In questo caso guardando la tabella in figura 49a la nostra scelta per l’estrazione dei dati ricade sulla combinazione **GATE + Tika** poiché è la combinazione più veloce per l’analisi di documenti web qualsiasi (non solo HTML).

La validazione dei risultati del nostro progetto si baserà su un set di dati prestabilito, in questo caso circa 100000 pagine web “qualunque” da cui sono stati realizzati circa 60000 InfoDocument/GeoDocument con controllo della lingua, e dai quali sono stati infine estrapolati 60 InfoDomainDocument/GeoDomainDocument relativi a domini web differenti.

I set di pagine web sono stati presi in modo casuale. Divideremo la validazione delle pagine web in quattro categorie per avere un struttura matricale R:[true,false] C:[true,false] dove tali valori sono determinati dal confronto positivo e negativo da parte dell’ utente e del nostro “Robot” il modo con cui ci riferiremo al progetto di questo studio:

* **Confronto positivo sia dal Robot, sia dall'utente**: vuol dire che le informazioni estratte dal nostro Robot identificano un luogo geografico che è lo stesso percepito anche dall’utente umano che ha letto la pagina.
* **Confronto negativo da parte dell’Utente**: le informazioni estratte dal Robot identificano un luogo che non è lo stesso identificato dall’utente, è necessario però fare un’osservazione, il nostro Robot può essere in parte più intelligente dell’Utente infatti:
  1. L’utente che legge la pagina web percepisce il luogo di interesse come *Polo Scientifico e Tecnologico* mentre il progetto (con l’ausilio di Google Maps), indica le informazioni di Geolocalizzazione del *Polo Scientifico e Tecnologico di Sesto Fiorentino* che è un’informazione più dettagliata ed efficiente, grazie a informazioni aggiuntive che l’Utente non percepisce in lettura e a meta-informazioni della pagina web che l’utente non può fisicamente leggere.
  2. Succede anche che il Robot identifichi una informazione meno dettagliata rispetto a quella percepita dall’Utente per esempio verifica la città di Firenze con le rispettive coordinate mentre l’utente percepisce altre informazioni che specificano meglio la località per esempio un particolare zona di Firenze.
* **Confronto negativo da parte del Robot***:* Il Robot non riesce a estrarre informazioni di alcun genere dalla pagina web mentre l’utente si.
* **Confronto, negativo sia per il Robot sia per l'utente :** Sia il Robot che l’utente falliscono nella ricerca d’informazioni per esempio la pagina web di registrazione a un sito nn contiene informazioni di alcun genere sulla geolocalizzazione dell’azienda del sito.

Vediamo un campione dei record della tabella GeoDomainDocument relativi a dei GeoDocument che andremo ad analizzare:

(1, 'www.apretoscana.org', 'Toscana', 43.8320966, 11.2042694),

(2, 'www.unifi.it', 'Firenze', 43.9303475, 10.9078587),

(3, 'www.cism.unifi.it', 'Firenze', 43.7228386, 10.4016888),

(4, 'www.lasis.unifi.it', 'Firenze', 43.7710332, 11.2480006),

(5, 'www.cem.unifi.it', 'Firenze', 43.7710332, 11.2480006),

(6, 'www.unisi.it', 'Siena', 43.3191812, 11.3327523),

(7, 'www.biblio.unipi.it', 'Pisa', 43.7228386, 10.4016888),

(8, 'www.bpnet.apretoscana.org', 'Florence', 43.8320966, 11.2042694),

(9, 'www.lenst.det.unifi.it', 'Firenze', 43.7710332, 11.2480006),

(10, 'www.unipi.it', 'Pisa', 43.7228386, 10.4016888),

(11, 'www.lam.unifi.it', 'Firenze', 44.6989932, 10.6296859),

(12, 'www.csdc.unifi.it', 'Firenze', 39.2230155, 9.110908799999999),

(13, 'www.emc.unifi.it', 'Firenze', 43.7710332, 11.2480006),

(14, 'www.ows.ing.unifi.it', 'Firenze', 25.5130285, -103.3828327),

……………………………………………………………………………….

**ATTENZIONE:** Nelle successive tabelle di validazione andiamo a indicare il Confronto negativo sia da parte del Robot che dell’utente umano con “Missing” in questo caso si intende il fatto che l’informazione è “*mancante*” e non viene vista da entrambi, la “*corretezza*” dell’informazione se trovata invece è un’altra cosa, se l’informazione nella pagina web in esame è presente, ma sbagliata, per esempio “via telematica 6” sembrerebbe in effetti un possibile indirizzo fisico sia dal punto di vista del robot che dell’essere umano , in realtà seguendo il contesto del contenuto del testo della pagina web ci si accorge che l’informazione è sbagliata perchè non riferita a un’indirizzo fisico, nel dataset utilizzato per i test più della metà delle pagine fanno riferimento a pagine di registrazione, immagini e altre pagine web di questo genere che difficilmente avranno informazioni come numero di telefono e indirizzo che sono quelle che cerchiamo di estrarre.

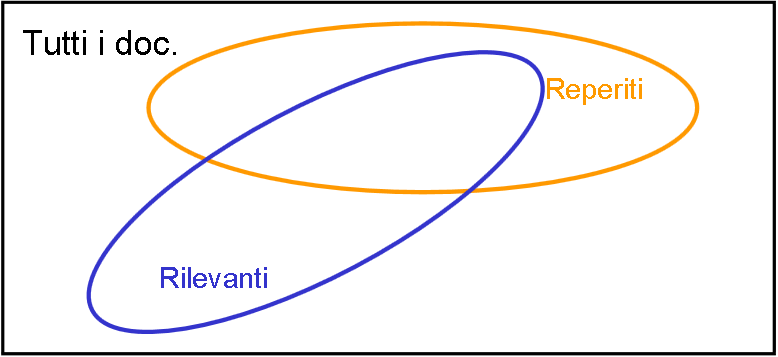
Per mostrare una giusta prospettiva del funzionamento del nostro Robot vengono fatte due validazioni:

1. Una sul dataset fornito considerando come “Confronto Positivo” da parte del Robot della informazione non corretta, (giustamente lui a trovato una porzione di testo che rispetta tutti i parametri forniti), mentre l’ente umano la considererà come “Confronto Negativo”, leggendo il documento dall’inizio alla fine si accorge che il testo fa riferimento ad altro.
2. Una su un dataset dei soli domini cioè delle pagine principali dei siti (Homepage,ecc.) che contengono sicuramente informazioni di geolocalizzazione come indirizzo o P.IVA. Qui si considera come “Confronto Negativo” da parte del Robot della informazione non corretta, poiché l’informzione “corretta” si presume esista per forza (può anche succedere che manchi del tutto).

Diamo una veloce introduzione a quali criteri utilizziamo per fornire un buon riferimento sull’efficienza del nostro programa/robot, dividiamo i documenti in due gruppi Rilevanti e Reperiti.

**Che cosa si valuta?** Funzionalità, Correttezza e Performance (sia di tempo che di spazio)

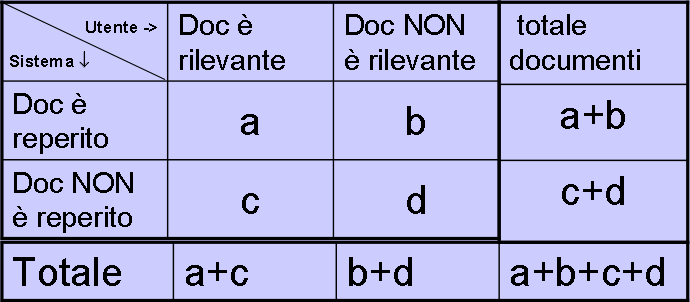
**Come si valuta?** Analisi Funzionale, Analisi degli Errori, Efficacia del retriveal (precisione e completezza dei documenti reperiti).



**Fig.49b: Rapp. grafica della distinzione di documenti.**

***L’obbiettivo per una buona efficacia del progamma è reperire quanti più documenti rilevanti possibile, minimizzando il numero di documenti non rilevanti reperiti.***

La metodologia di valutazione utilizzata in questo studio è la **Matrice di Confronto**:



(spesso c non è noto)

**Fig.49c:rappresentazione grafica generale di una matrice di confronto**

Viene introdotto un nuovo parametro di valutazione l’accuratezza (Accuracy), i cui problemi nei casi di collezzioni molto grandi come quella presa in considerazione derivano dal fatto che:

* La maggior parte dei documenti non sono rilevanti
* La maggior parte dei documenti non sono reperiti
* Si ha un’amplificazione non indifferente del valore di accurattezza a causa di *d>>a*

Strumenti più avanzati tendono a combinare Precision e Recall in un unico parametro attraverso una media armonica, in generale i valori di Precision e Recall sono calcolati a valori prestabiliti del totale 10%,20%,90%. In seguito viene svolta la media tra questi “livelli di richiamo” come sono chiamati spesso.

Queste sono sistemi di calcolo più avanzati, noi ci limiteremo a calcolare tali valori per un campione abbastanza piccolo del nostro dataset 150000 pagine web su circa 6000000.

Nella tabella che segue oltre al confronto incrociato tra utente umano e robot sono stati calcolati i vari criteri di valutazione per i sistemi di information retrieval (IR), Precisione (Precision), Sensitivita (Sensitivity o Recall), Accuratezza (Accuracy),ecc.

Tabella per la validazione del progetto relativa a 150000 URL da cui sono stati creati circa 50000 GeoDocument, da cui sono stati estratti circa 50 GeoDomainDocument con domini diversi.

I valori percentuali nella prima validazione sono stati arrotondati per difetto per facilitare la lettura.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | **Reference Condition** | |  | |
|  |  | **Utente Umano Confronto *Positivo*** | **Utente Umano Confronto *Negativo*** |  |
| **Test / System Output**  **(GATE + Tika)** | **Robot Confronto *Positivo*** | ***True Positive* (TP)** | ***False Positive* (FP)** | ***Precision*** = |
| **Robot Confronto *Negativo*** | ***False Negative* (FN)** | ***True Negative* (TN)** | ***Negative Predictive Value*** = |
|  |  | ***Sensitivity*** or ***Recall***= | ***Specificity***= | ***Accuracy***= |

**Fig.50: Rappresentazione della matrice di confronto utilizzata per questo studio.**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **AnnotationType** | **AnnotationSet** | **TP**  **(correct)** | **FP**  **(partiallycorrect)** | **FN & TN**  **(missing)** | **Mat. Conf.** | **Precision** | **Recall** | **Accuracy** |
| **MyLocalita** | **MyFOOTER** | **48413(96.8%)** | **1587(3.2%)** | **0(0%)** | TP=48413  FP=1587  FN=0  TN=0 | **0.96826** | 1 | **0.96826** |
| **MyLocalita** | **MyHEAD** | **0(0%)** | **0(0%)** | **50000(100%)** |
| **MyLocalita** | **MySpecialID** | **18211(36.4%)** | **31789(63.6%)** | **0(0%)** |
| **MyLocalita** | **MyAnnSet** | **48413(96.8%)** | **0(0%)** | **1587(3.2%)** |
| **MyIndirizzo** | **MyFOOTER** | **8000(16%)** | **5500(11%)** | **36500(73%)** | TP=8000  FP=5500  FN=12748  TN=23752 | **0.221876834** | **0.820960347** | **0.20414** |
| **MyIndirizzo** | **MyHEAD** | **0(0%)** | **0(0%)** | **50000(100%)** |
| **MyIndirizzo** | **MySpecialID** | **4210(8.4%)** | **35790(71.6%)** | **4000(8%)** |
| **MyIndirizzo** | **MyAnnSet** | **13783(27.6%)** | **6210(12.4%)** | **30007(60%)** |
| **MyEmail** | **MyFOOTER** | **47777(95.4%)** | **22(0%)** | **2201(4.4%)** | TP=47777  FP=22  FN=0  TN=2201 | **0.999539739** | 1 | **0.95554** |
| **MyEmail** | **MyHEAD** | **0(0%)** | **0(0%)** | **50000(100%)** |
| **MyEmail** | **MySpecialID** | **18189(36.4%)** | **11(0%)** | **31800(63.6%)** |
| **MyEmail** | **MyAnnSet** | **47777(95.4%)** | **22(0%)** | **2201(4.4%)** |
| **MyTelefono/MyFax** | **MyFOOTER** | **5308(4.3%)** | **0(0%)** | **44692(89.4%)** | TP=5308  FP=0  FN=0  TN=44692 | 1 | 1 | 1 |
| **MyTelefono/MyFax** | **MyHEAD** | **0(0%)** | **0(0%)** | **50000(100%)** |
| **MyTelefono/MyFax** | **MySpecialID** | **18200(36.4%)** | **0(0%)** | **31800(63.6%)** |
| **MyTelefono/MyFax** | **MyAnnSet** | **5308(4.3%)** | **0(0%)** | **44692(89.4%)** |
| **MyEdificio** | **MyFOOTER** | **43902(87.8%)** | **1799(3.6%)** | **4299(8.6%)** | TP=43902  FP=1799  FN=4187  TN=112 | **0.960635434** | **0.912932271** | **0.87804** |
| **MyEdificio** | **MyHEAD** | **38605(77.25%)** | **0(0%)** | **11395(22.75%)** |
| **MyEdificio** | **MySpecialID** | **18207(36.4%)** | **0(0%)** | **31793(63.6%)** |
| **MyEdificio** | **MyAnnSet** | **11500(23%)** | **33500(67%)** | **5000(10%)** |
| **MyPartitaIVA** | **MyFOOTER** | **5900(11.8%)** | **0(0%)** | **44100(88.2)%** | TP=49100  FP=0  FN=900  TN=0 | 1 | **0.982** | **0.982** |
| **MyPartitaIVA** | **MyHEAD** | **0(0%)** | **0(0%)** | **50000(100%)** |
| **MyPartitaIVA** | **MySpecialID** | **18200(36.4%)** | **0(0%)** | **31800(63.6%)** |
| **MyPartitaIVA** | **MyAnnSet** | **900(1.8%)** | **0(0%)** | **49100(98.2)%** |
| **TOTALE delle pagine aventi come minimo indirizzo,edificio e localita ovvero le informazioni principali (i numeri delle pagine sono arrotondati)** | | **11000(22%)** | **38000(76%)** | **6000(12%)** | TP=11000  FP=38000  FN=6000  TN=0 | **0.224489795** | **0.647058823** | **0.22** |

**Fig.51: Tabella della validazione relativa a 50000 url, per le singole annotazioni di GATE e ogni singolo AnnotationSet di Interesse. Le percentuali dei valori sono state arrontondate e in certi casi sono uguali semplicemente perché prendono lo stesso testo.**

I risultati della validazione “crollano” per il tracciamento errato dell’indirizzo da parte del robot e per i troppi dati “mancanti” dalle pagine web fornite dal dataset, per verificare che il nostro Robot sia efficiente viene svolta una seconda validazione sulle pagine principali dei domini presenti nel dataset (circa 500) quindi con sicuramente dei dati come l’indirizzo e la partita iva a dispetto di prima dove la validazione si basa in gran parte su pagine di registrazione e simili.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **AnnotationType** | **AnnotationSet** | **TP**  **(correct)** | **FP**  **(partiallycorrect)** | **FN & TN**  **(missing)** | **Mat. Conf.** | **Precision** | **Recall** | **Accuracy** |
| **MyLocalita** | **MyFOOTER** | **500(100%)** | **0(0%)** | **0(0%)** | TP=500  FP=0  FN=0  TN=0 | 1 | **1** | **1** |
| **MyLocalita** | **MyHEAD** | **125(25%)** | **0(0%)** | **375(75%)** |
| **MyLocalita** | **MySpecialID** | **0(0%)** | **0(0%)** | **500(100%)** |
| **MyLocalita** | **MyAnnSet** | **500(100%)** | **0(0%)** | **0(0%)** |
| **MyIndirizzo** | **MyFOOTER** | **325(65%)** | **65(13%)** | **110(22%)** | TP=325  FP=65  FN=0  TN=110 | **1** | **1** | **0.87** |
| **MyIndirizzo** | **MyHEAD** | **0(0%)** | **0(0%)** | **500(100%)** |
| **MyIndirizzo** | **MySpecialID** | **425 (85%)** | **50(10%)** | **25(5%)** |
| **MyIndirizzo** | **MyAnnSet** | **300(60%)** | **75(15%)** | **125(25%)** |
| **MyEmail** | **MyFOOTER** | **340(68%)** | **0(0%)** | **160(32%)** | TP=340  FP=0  FN=0  TN=160 | 1 | 1 | 1 |
| **MyEmail** | **MyHEAD** | **0(0%)** | **0(0%)** | **500(100%)** |
| **MyEmail** | **MySpecialID** | **500(100%)** | **0(0%)** | **0(0%)** |
| **MyEmail** | **MyAnnSet** | **500(100%)** | **0(0%)** | **0(0%)** |
| **MyTelefono/MyFax** | **MyFOOTER** | **370(100%)** | **0(0%)** | **130(26%)** | TP=370  FP=0  FN=0  TN=130 | 1 | 1 | 1 |
| **MyTelefono/MyFax** | **MyHEAD** | **0(0%)** | **0(0%)** | **500(100%)** |
| **MyTelefono/MyFax** | **MySpecialID** | **370(100%)** | **0(0%)** | **130(26%)** |
| **MyTelefono/MyFax** | **MyAnnSet** | **370(100%)** | **0(0%)** | **130(26%)** |
| **MyEdificio** | **MyFOOTER** | **454(91.88%)** | **33(6.6%)** | **13(2.6%)** | TP=454  FP=33  FN=13  TN=0 | **0.932238193** | **0.97216274** | **0.908** |
| **MyEdificio** | **MyHEAD** | **500(100%)** | **0(0%)** | **0(0%)** |
| **MyEdificio** | **MySpecialID** | **182(36.4%)** | **0(0%)** | **318(63.6%)** |
| **MyEdificio** | **MyAnnSet** | **115(23%)** | **335(67%)** | **50(10%)** |
| **MyPartitaIVA** | **MyFOOTER** | **256(51%)** | **0(0%)** | **244(49%)** | TP=256  FP=0  FN=0  TN=244 | 1 | 1 | 1 |
| **MyPartitaIVA** | **MyHEAD** | **0(0%)** | **0(0%)** | **500(100%)** |
| **MyPartitaIVA** | **MySpecialID** | **256(51%)** | **0(0%)** | **244(49%)** |
| **MyPartitaIVA** | **MyAnnSet** | **0(0%)** | **0(0%)** | **500(100%)** |
| **TOTALE delle pagine aventi come minimo indirizzo,edificio e localita ovvero le informazioni principali** | | **440(88%)** | **5(1%)** | **55(11%)** | TP=440  FP=5  FN=25  TN=30 | **0.988764044** | **0.946236559** | **0.94** |

**Fig.51b: Tabella della validazione relativa a 500 url di domini web, per le singole annotazioni di GATE e ogni singolo AnnotationSet di Interesse. Le percentuali dei valori sono state arrontondate e in certi casi sono uguali semplicemente perché prendono lo stesso testo.**

**Conclusione**

Questo studio è nato con lo scopo di sperimentare l'efficacia di un sistema di classificazione in grado di estrarre delle informazioni di geolocalizzazione dal testo delle pagine web relativo all’argomento descritto in esso.

La difficoltà di tale realizzazione è motivata dalla tipologia dei documenti stessi: essi sono stati estratti dal web e si presentano in forma di documenti di testo non strutturato e scritti in linguaggio umano, usando una terminologia umana non standard e priva di struttura che li renda facilmente processabili.

L'idea di affrontare questa problematica nasce dall'esigenza di riuscire a manipolare i dati, sempre più numerosi, presenti nel web , strutturati in giganteschi database caratterizzati da rigide modalità di accesso e di querying, abbiamo visto in modo superficiale Google Maps che ha potenti query di ricerca dell’informazione.

La quantità di dati disponibili è considerevole, lo studio ha evidenziato le problematiche relative allo stile e alla lingua con cui sono scritti, l'uso di un vocabolario non controllato e la mancanza di parole frequenti

utili alla classificazione. Per tentare di superare queste difficoltà, si è ricorsi al Natural Language Processing, utile per estrarre, mediante analisi lessicale e sintattica l’informazione ritenuta utile allo scopo prefissato. In questa fase abbiamo usato un applicativo open source, GATE che pur non essendo predisposto per l’analisi di testi scritti in lingua italiana, si è rivelato utile per estrarre entità e relazioni sotto forma di annotazioni, che pur non aggiungendo valore semantico alle informazioni dei documenti web ne arricchiscono il contenuto.

Il risultato ottenuto impostando determinati parametri, attributi e vincoli sul dataset considerato ha permesso di dimostrare le potenzialità di questo metodo per l'esecuzione su grosse quantità di dati.

Per questo studio la sfida futura come in molti altri ambienti del web , è riuscire a strutturare ulteriormente l'informazione creando ontologie di dominio o addirittura ontologie superiori mediante le quali concettualizzare una particolare conoscenza in modo univoco, migliorare l'accuratezza delle ricerche, superare le differenze della terminologia, agevolare l'interoperabilità dei sistemi informatici e inferire relazioni che, in certi casi, possono rivelarsi apportatrici di nuova informazione. In un simile ambiente, è impensabile un intervento “manuale” diretto nella strutturazione e manipolazione dei dati: Information Extraction, Natural Language Processing e Machine Learning rappresentano dunque un contributo importante in questo processo evolutivo, e la classificazione automatica dei referti ottenuta ne è la dimostrazione.

**APPENDICE**

1. **Introduzione veloce all’utilizzo dei file .par**

Un genere di file poco conosciuti in ambiente informatico sono i file con estensione .par in questo paragrafo fornirò una veloce introduzione a questo genere di file e al loro utilizzo nell’ambito del nostro progetto

**Che cosa è un file .par???**

In parole povere un file compresso come .zip o .rar, ma con la capacità in caso di deterioramento dei dati di rintracciare in rete le copie dei dati corrotti e utilizzarli al loro posto, di solito sono usati da coloro che lavorano nel ramo degli usernet.

A cosa ci servono nel nostro progetto? Per motivi che elencheremo ne l paragrafo 7 , essi sono i file contenenti la nostra “personale” grammatica cioè se visto come un file di testo è un insieme gigantesco di parole inerenti a una particolare lingua e in tutte le varie forme ad esempio per la lingua italiana si potrebbe avere: *dovere dovrei dovrebbe dovuto ecc.*

**Si poteva utilizzare un file .zip o .rar o altro genere?**

In teoria si, ma quando si lavora a un progetto già iniziato come sto facendo io, non si ha il lusso di scegliere quello che ci viene fornito.

**Osservazione**: La soluzione perfetta visto che utilizzia GATE/JAPE sarebbe stato avere la nostra grammatica in formato .gapp, o .xgapp che è quella utilizzata di default dal programma.

1. **Link Utili alla documentazione GATE**

La documentazione ufficiale è disponibile al seguente indirizzo:

<http://gate.ac.uk/>

Consiglio inoltre di dare un’occhiata hai seguenti link sempre nel sito di GATE:

<http://gate.ac.uk/sale/tao/>

<http://gate.ac.uk/wiki/training-materials-2011.html>

<http://gate.ac.uk/demos/gate-tutorial-all/part-one-all-with-annic-and-ontology.html>

<http://gate.ac.uk/demos/developer-videos/>

Consiglio vivamente di provare il tutorial al seguente indirizzo completo di tutte le informazioni e screenshoot passo a passo per l’utilizzo di GATE:

<http://gate.ac.uk/wiki/quick-start/>

Documentazione sulle regole JAPE

<http://gate.ac.uk/sale/tao/splitch8.html#chap:jape>

http://gate.ac.uk/sale/thakker-jape-tutorial/GATE%20JAPE%20manual.pdf

Lista completa delle regole JAPE di default per GATE

<http://gate.ac.uk/sale/tao/splitap6.html>

L’unica pecca nella documentazione è che per chi è nuovo dell’argomento non è spiegato molto bene come customizzare (modificare per proprio conto) gli elementi della pipeline del GATE .

1. **Programmazione delle regole JAPE[[39]](#footnote-39)**

JAPE consente di riconoscere le espressioni regolari nelle annotazioni sui documenti .

Le prime righe di ciascuna grammatica costituiscono l'head e contengono il nome della fase (namePhase), gli input (Lookup), le modalità di match (control e debug). L'input indica il tipo di dati, ottenuti da precedenti elaborazioni, su cui si vogliono eseguire delle azioni in particolare.Andiamo a vedere le varie componenti di una regola JAPE nel dettaglio:

**Phase: namePhase //nome della fase**

**Input: Lookup //input della regola**

**Options: control=appelt debug=true //opzioni di controllo delle regole.**

**Rule: nameRule //nome della regola**

**Priority: 10 //priorità della regole**

**( <PATTERN>) //Pattern della Regola Jape nota anche con l'acronimo LHS**

**:temp //fine LHS parte sinistra**

**--> //dalla sinistra passo alla destra**

**:temp.call\_name //inizio RHS parte destra**

**{ JAVA BLOCK } //blocco di codice JAVA nota anche con l'acronimo RHS**

**Fig.52: Struttura di una regola JAPE**

* **Phase**: identifica a quale fase appartiene la regola, una fase può avere più regole il cui ordine è determinate dalle opzioni di controllo e dalle priorità.
* **Senza Input**: Se non è previsto input allora si prendono solo le annotazioni di default (LookUp,Token,ecc.)del GATE, cioè si può riutilizzare solo le annotazioni delle regole JAPE già definite di default dal GATE.
* **Con Input:** Con elementi di INPUT nostri ci sono 2 modi di lavorare:
  + - O si crea delle nuove annotazioni basate su quelle di default in questo modo non c’è bisogno di aggiungere o togliere niente al Gazzeter di default ANNIE.
    - andiamo a introdurre delle liste esterne al nostro Gazzeter di GATE , il tutorial a <http://gate.ac.uk/wiki/quick-start/> ne mostra un chiaro esempio.
* **Priority(priorità)**: abbastanza intuitivo, quando ci sono più regole che lavorano su una stessa fase. Ci sono 3 regole a seguire:
  + - Da tutte le regole corrispondenti una regione del documento di partenza ad un certo punto X, quella che corrisponde alla regione più lunga è lanciata per prima o in modo esclusivo.
    - Se più di una regola corrisponde alla stessa regione, quello con la più alta priorità viene lanciata per prima o in modo esclusivo.
    - Se vi è più di una regola con la stessa priorità e che lavorano nella stessa zona, quella definita per prima nella sequenza delle fasi è lanciata per prima o in modo esclusivo.

*Per impedire l’esecuzione delle regole o posticiparle ed evitare conflitto tra le fasi si utilizza il controllo delle OPTIONS*.

* **Options : Essa a 2 elementi Control e Debug**
  + **Control**: Per il matching o controllo sono previste cinque modalità di esecuzione

1. ***Brill***: Se una o più regole individuano lo stesso match, sono tutte attivate contemporaneamente. Si avrà pertanto la creazione di più annotazioni all'interno di una stessa fase sulla medesima porzione di testo. Se più regole trovano match con uguale posizione iniziale ma lunghezza diversa, la ricerca di match proseguirà a partire dalla posizione in cui finisce il match più lungo.
2. ***All***: È simile a brill ma la regola prosegue la ricerca di match a partire dalla fine del match corrente.

***Brill and All sembrano praticamente uguali,*** *entrambi vanno a lanciare tutte le regole e nello stesso ordine ma l’opzione all fa una specie di viaggio di ritorno attraverso le regole. Andiamo a vederli più nel dettaglio, dato un esempio dove [] sono annotazioni con la nostra regola che ha la struttura del tipo (stiamo lavorando nella parte LHS):* ***[aaa[bbb]] [ccc [ddd]]****. Se ci sono due regole ,* ***[aaa[bbb]]*** *e* ***[ccc [ddd]]****che lavorano nella stessa regione/zona di interesse entrambe vengono eseguite ma in ordine diverso (l'ordine delle regole dipende dalla priorità):*

*BRILL:* ***[aaabbb] [cccddd]*** *esegue per esempio le regole rule3,rule2,rule1*

*ALL:* ***[aaa[bbb]] [ccc[ddd]]*** *esegue le regole rule3,rule2,rule1,rule2,rule3*

1. ***First***: Appena una regola trova un match si attiva senza curarsi di un eventuale match più lungo
2. **Once**: Appena una regola è stata attivata, l'intera fase di JAPE si conclude dopo il primo match
3. ***Appelt***: Solo una regola può essere attivata nella stessa regione di testo, secondo precise regole di priorità:
   1. tra tutte le regole che trovano un match con posizione iniziale identica, si attiva solo quella che corrisponde al match più lungo.
   2. se una o più regole trovano un match su una stessa porzione di documento, si attiva quella con priorità più alta.
   3. se c'è più di una regola con la stessa priorità, si attiva quella definita per prima

* **Debug:** gli viene assegnato un valore booleano se è true viene utilizzato per visualizzare i messaggi di conflitto sul display di standard output.
* **temp:** nome temporaneo della regola essa a valore solo all'interno della regola stessa
* **call\_name:** nome assegnato alla regola, una sovrascrittura al nome temporaneo, serve per poter richiamare la regola già definita da altri file JAPE facenti parte della stessa sequenza multifase.

Con riferimento alla modalità appelt, si può dichiarare un parametro opzionale di priorità associato a ciascuna regola che di solito è un numero intero positivo. Un numero più alto rispetto ad un altro corrisponde ad una priorità maggiore. Se la priorità non è dichiarata, per default tutte le regole hanno priorità -1. La quarta riga dichiara il nome della prima regola: la presenza di più regole in una fase è consentita ma è bene fare attenzione all'ordine con cui sono scritte ed eseguite per evitare risultati inattesi.

La quarta riga dichiara il nome della prima regola: la presenza di più regole in una fase è consentita ma è bene fare attenzione all'ordine con cui sono scritte ed eseguite per evitare risultati inattesi.

Subito dopo aver specificato il nome della regola, si distinguono due parti: Left-Hand Side (LHS) e Right-Hand Side (RHS), separate dalla sequenza “-->”.

La LHS della regola specifica i pattern che devono essere trovati (matched) nel documento. Si noti che nella LHS si dichiarano come match da trovare annotazioni già esistenti ottenute da fasi precedenti a quella in cui è presente la regola in corso di esecuzione. Tramite un'opportuna sintassi, si può specificare un match che tiene conto anche degli attributi associati all'annotazione e dei relativi valori.

Se un'annotazione viene specificata in una LHS ma per errore non viene indicata nell'input, la regola sarà ignorata. Ogni annotazione da trovare è racchiusa in parentesi graffe mentre il pattern da individuare è racchiuso in parentesi tonde e associato ad una variabile. Spesso si è rivelato utile ricorrere a operatori contestuali:

**(**

**/\*\*Cerca tutti i nomi delle regioni "contenuti" nei Sentence del documento\*/**

**{Sentence contains VarRegioni}**

**/\*\*Cerca tutti i nomi delle regioni "all'interno" dei Sentence del documento\*/**

**{VarRegioni within Sentence}**

**)**

**Fig.53:Esempio con gli operatori contestuali**

In questo caso l’operatore **contains** e **within** sono l’uno il complementare dell’altro, le due annotazioni sopra citate identificano le stesse informazioni nel documento.

Quando non si è in grado di definire a priori il numero di occorrenze di un'annotazione da individuare, si può ricorrere agli operatori di unione e Kleene particolarmente utili e frequentemente usati:

**Pattern elements possono essere combinati in vari modi**

* Sequencing: **{Token}{Token}**
* Alternatives: **{Token} | {Lookup}**
* Grouping with parentheses: **({Token}{Token})**

**Espressioni regolari e operatori di molteplicità**

* zero-o-uno: **({MyAnnot})?**.
* zero-o-più **({MyAnnot})\***.
* uno-o-più: **({MyAnnot})+**.
* esattemente n: **({MyAnnot})[n]**.
* tra n e m (inclusive): **({MyAnnot})[n,m]** .
* **NOTA da programmatore**: in generale se siete programmatori avanzati del settore saprete che i simboli **?,\*,+** sono deprecati a favore delle loro alternative equivalenti [0,1][0,n][1,n] semplicemente perché è stato dimostrato che vengono elaborate più in fretta e sono anche più precise.
* **{Token}** la parola più generale possibile.
* La lunghezza del token o numero di caratteriè specificata dalla feature “length” **{Token.length > 4}** ,supporta anche **<, <=, >=, !=** .
* Le regular expressions **=~, ==~,!~, !=~.** e.g {Token.string =~ "[Dd]i"} identifica sia “di” che “Di”.
* Specificare una stringa di testo e.g **{Token.string == "di"}** identifica la stringa “di” nel testo.
* Specificare una annotazione precedentemente assegnato da un dizionario geografico, tokeniser, o altro modulo e.g **{Lookup}** le liste o dizionai del Gazetteer di GATE hanno a disposizione tre feature di default che è possibile richiamare con tale comando: **{Lookup.minorType == nomeLista},{Lookup.majorType == nomeLista},{Lookup.language == nomeLista}** è possibile addirittura utlizzare un altro nome per l’annotationType del Gazetter per esempio “Control” e quindi **{Control.majorType}**.
* Specificare gli attributi di una annotazione (kind):
  + **{Token.kind == number}** : prende tutti i Token che corrispondono a un numero.
  + **{Token.kind == word}**: prende tutti i token che corrispondono a parole.
  + La cosa vale anche per altre annotazionee.g. **{SpaceToken.kind == space}.**
* Specificare l’ortografia dell’ annotazione (orth ):
  + **{Token.orth == upperInitial }**: la lettera iniziale è maiuscola e le alter sono minuscole.
  + **{Token.orth == allCaps}**: le lettere sono tutte maiuscole.
  + **{Token.orth == lowerCase}** : le lettere sono tutte minuscole.
  + **{Token.orth == mixedCaps}** : le lettere sono sia maiuscole che minuscole.
* Operatori logici:
  + **OR (|) :** e.g.({Token}|{SpaceToken}) .
  + **AND(,) :** e.g.({Token,SpaceToken}).

**Per quanto riguarda il blocco JAVA della regola JAPE**

* Per impostazione predefinita, ogni azione a le seguenti classi importate: java.io.\*, java.util.\*, gate.\*,gate.jape.\*, gate.creole.ontology.\*, gate.annotation.\*, and gate.util.\*.
* **Named Java Blocks:** È possibile assegnare un blocco di Java con una etichetta dal LHS. Il blocco viene chiamato solo se vi è almeno una annotazione legata alla etichetta .All’interno del blocco Java c'è un labelAnnots variabile riferiti alla AnnotationSet legato all'etichetta

*ie AnnotationSet xyAnnots = bindings.get ("xy").*

* **Java Block Parameters:**I parametri disponibili per i blocchi Java RHS sono:
  + **doc** Il documento attualmente in fase di elaborazione.
  + **inputAS** Il *AnnotationSet* specificato dalparametro runtime

*inputASName* al JAPE trasduttore di PR. Leggere o cancellare annotazioni da qui.

* + **outputAS** Il *AnnotationSet* specificato dal parametro runtime

*outputASName* al JAPE trasduttore di PR. Creare nuove annotazioni qui.

* + **ontology** L'ontologia (se del caso) fornito come parametro di runtime per il PR trasduttore JAPE.
  + **bindings**: permette di gestire la binding map….
* Le RHS di una regola JAPE possono avere qualsiasi numero di*: bind.Type = {}* espressioni di assegnamento e blocchi di Java codice, separati da virgole.
* **Gestione delle eccezioni:** 
  + Qualsiasi *JapeException o RuntimeException* lanciata da un blocco Java RHS farà fallire il traduttore JAPE PR con un *ExecutionException*
  + Per gli errori non fatali in un blocco di RHS si può buttare un *gate.jape.NonFatalJapeException*
  + Questo stamperà le informazioni di *debug* se impostato a true(nome di fase, nome di dominio, il file e il numero di riga), ma non si interrompe l'esecuzione del trasduttore.
    - Tuttavia interromperà questa regola, cioè se vi è più di un blocco o di un’assegnazione sul RHS, quelli dopo il *throw* non sarà eseguiti.
* **Returning from RHS blocks:** Si può far ritornare (return) da un blocco Java RHS, che previene eventuali blocchi successivi o assegnazioni per l'esecuzione di tale regola .

Come primissima regola che vedremo prenderò la regola Jape email.jape[[40]](#footnote-40) una di quelle fornite di default da GATE[[41]](#footnote-41) essa serve per il tracciamento delle email all’interno di un testo, o meglio di tutte le keyword che hanno una struttura uguale a quella di un indirizzo email, e.g corroborante@diesel.it non è un indirizzo email ma viene tracciata comunque:

**// Email rules di default delplguin ANNIE**

**Rule:Emailaddress1**

**Priority: 50**

**(**

**({Token.kind == word}|{Token.kind == number} )+**

**({Token.string == "\_"} )?**

**({Token.string == "."})?**

**({Token.kind == word}| {Token.kind == number}| {Token.string == "\_"} )\***

**{Token.string == "@"}**

**({Token.kind == word}| {Token.kind == symbol}|{Token.kind == punctuation}|{Token.kind == number})**

**({Token.string == "."})?**

**({Token.kind == word}|{Token.kind == symbol}|{Token.kind == punctuation}|{Token.kind == number} )\***

**({Token.string == "."})?**

**({Token.kind == word}|{Token.kind == symbol}| {Token.kind == punctuation}| {Token.kind == number} )?**

**({Token.string == "."})?**

**(**

**{Token.string == "."}**

**({Token.kind == word}| {Token.kind == number})**

**({Token.string == "."})?**

**( {Token.kind == word}| {Token.kind == number})?**

**({Token.string == "."})?**

**( {Token.kind == word}| {Token.kind == number} )?**

**)**

**)**

**:emailAddress --> :emailAddress.Email= {kind = "emailAddress", rule = "Emailaddress1"}**

**Fig.54:Regola email.jape di default di GATE**

**Se siete riusciti a leggere la precedente regola Jape, siete pronti a creare le vostre prime regole Jape.**

Abbiamo detto che ogni regole JAPE è identificata da una fase che può contenere più regole, che si gestiscono con priorità e opzione di controllo, ma non tutto può essere scritto in un unico file JAPE , nasce quindi l’esigenza di gestire l’ordine in cui vengono eseguiti i file JAPE stessi. Una volta create tutte le nostre regole nei file JAPE creiamo un file **main.jape**, esso semplicemente è come la radice di un’albero i cui rami sono i file JAPE identificate da una fase, e le foglie di ogni ramo le regole JAPE rispettivamente associate. Il concetto di **Multifase** deriva dal comando che gli viene dato in prima riga di comando. **Esso sarà l’unica regola JAPE che passeremo alla pipeline del GATE.** **Attenzione: Le regole JAPE richiamate dal main devono per forza essere nella stessa directory/Cartella**.Vediamo un’esempio di file main.jape:

**MultiPhase:RelationExtractionGrammars**

**Phases:**

**myPreprocessing0**

**myPreprocessing1**

**myGeolocation**

**myIndirizzo**

**myPartitaIVA**

**myEmail**

**myTelefono**

**Fig.55: esempio di un file main.jape per caicare più regole jape una dietro l’altra.**

E’ in pratica una lista dei file .jape su cui vogliamo andare a lavorare, nella stessa cartella ci sono regole jape non chiamate dal main.jape non succede niente semplicemente rimangono lì il main.jape non sa della loro esistenza.

Le Macro, nelle regole Jape sono veri e propri pezzi di pattern LHS riutilizzabili da tutte le regole successive in tutte le fasi successive devono essere invocate dopo gli Input e le Options ma prima della regola Jape (Rule) su cui si vuole utilizzarle.

La struttura è molto semplice ***MACRO: nome\_macro(blocco\_LHS*)** ,vediamone un’esempio:

**Phase: Indirizzo**

**Input: Token Lookup Split SpaceToken localita indirizzo provincie MyProvincia regioni industriale azienda assicurazione**

**articoloIndeterminativo articoloDeterminativo preposizioni avverbiDiLuogo locuzioni congiunzione trash**

**Options: control=all**

**MACRO:CONJ\_ITA(**

**(**

**{Lookup.majorType==articoloIndeterminativo} /\*(1)un una uno un'\*/**

**|{Lookup.majorType==articoloDeterminativo} /\*(2)il lo l' la i gli le\*/**

**|{Lookup.majorType==preposizioni} /\*(3)a agli ai al all' alla alle allo assieme attraverso \*/**

**|{Lookup.majorType==avverbiDiLuogo} /\*(4)altrove ci codesto costà costì dappertutto davanti dentro\*/**

**|{Lookup.majorType==locuzioni} /\*(5)accanto a vicino a davanti a innanzi a lontano da invece di insieme con\*/**

**|{Lookup.majorType==congiunzione} /\*(6)altresì ancora anzi ciò nonostante\*/**

**)**

**)**

**Rule : RuleMacro**

**(**

**(CONJ\_ITA)**

**):ita -->**

**:ita.Support={kind = “ita” , rule = RuleMacro}**

**Fig.56:Esempio con l’operatore MACRO**

1. **Customizzazione della PIPELINE di GATE**

Come premessa i seguenti metodi per la customizzazione sono stati fatti in ambiente **Windows 8 e alla versione GATE 7.1** non è detto che con altri sistemi o innovazioni di GATE le relative path ai vari file e plugin varino ,(per esempio nella versione 7.1 il plugin Tree\_Tagger è sparito ed è stato compresso insieme ad altri nel nuovo plugin Tagger\_Framework).

In questo capitolo andremo a descrivere come modificare in modo opportuno i determinati elementi PR della PIPELINE di GATE per una specifica lingua in questo caso l’italiano che non è fornita di default, per far questo è necessario seguire i seguneti passi:

1. Quando si crea un sistema di IE di nuovi linguaggi, è più facile iniziare con ANNIE e poi lavorare su ciò che deve adeguare .
2. Controllare le risorse in GATE per la lingua (se presente)
   1. Controllare la directory GATE\_plugins (suggerimento: la lingua plugins iniziano con Lang\_\*)
   2. Controllare il manuale d'uso per cose come POS Tagger che hanno varie opzioni di lingua
3. Verificare quali PR è possibile riutilizzare direttamente da ANNIE
   1. Tokeniser e Sentence Splitter i lavorano per la maggior parte delle lingue occidentali. Le lingue asiatiche possono richiedere componenti speciali.
4. Raccogliere tutte le altre risorse per la vostra lingua, ad esempio POS tagger. Questi possono essere implementati come plugin GATE.

Possiamo concettualmente dividere le PR in 3 tipologie, a seconda della complessità di customizzazione per elaborare informazioni testuali in altre lingue:

* **Language-Indipedent**(indipendente dal linguaggio):può non essere modificato per niente lavora con molte lingue con modifiche minime o nulle .
* **Easily Modifiable**(facilmente modificabile): può essere facilmente modificato per impostare una diversa lingua anche avendo poca esperienza di programmazione .
* **Language-Dipedent**(dipendono dalla lingua): possono essere modificati in modo semplice o complesso a seconda di quanto dobbiamo variare il PR in questione, si può arrivare addirittura a sostituirlo completamente con un nuovo PR.



**Fig.57: Immagine presa dalle slide IE Advanced, University of Sheffield, NLP**

**Language-Indipedent**

I PR di ANNIE che sono totalmente indipendenti dal linguaggio sono i *Document Reset* e gli *Annotazioni Set Transfer,* i quali possono essere visti come "*language-agnostic*", visto che fanno uso di annotazioni esistenti, senza riferimento al documento stesso o alla lingua utilizzata. Il *Tokeniser* e il *Sentence Splitter* sono (più o meno) *language-independent* e possono essere riutilizzati per lingue che hanno la stessa struttura semantica delle frasi (lo spazio bianco, punti fermi, ecc) . Assicurarsi di utilizzare l’ *Unicode Tokenizer*, non *l’English tokeniser*(che è personalizzato con alcune abbreviazioni inglesi, ecc.), qualche ritocco potrebbe essere comunque necessario , queste PR sono facili da modificare anche senza utilizzare il codice java.

**Easily Modifiable**

*Gazetteer* invece normalmente dipendono dalla lingua, ma possono essere facilmente tradotti o sostituiti con elenchi equivalenti trovati o generati , elenchi del telefono, giorni della settimana possono essere tradotti invece l’elenco delle città può essere trovato sul web. La modifica del Gazetteer non richiede alcuna programmazione o competenze linguistiche particolari.   
L’*Orthomatcher* lavorerà per altre lingue in cui applicano regole simili, ad esempio John Smith -> Mr Smith ,qui invece potrebbe essere necessario modificare a livello JAVA alcune variabili sono richieste anche competenze e conoscenze linguistiche.

**Language-Dipedent**

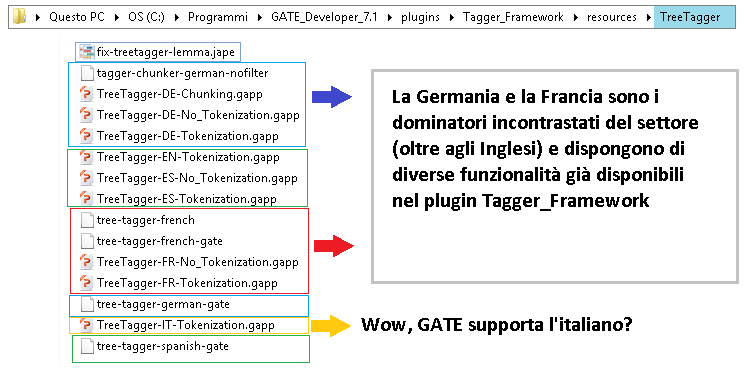
I *POS Tagger* e le *grammatiche* sono altamente dipendenti dalla lingua e al variare di stessa devono essre modificate anche di molto, se non esiste un *POS Tagger* per la lingua desiderata si può fare, un hack cioè una variante per un linguaggio specifico, infatti alcune regole grammaticali possono essere lasciate intatte, ma molte dovranno essere riscritte. Altre regole possono aver bisogno solo di piccole modifiche ad esempio l’ordine delle componenti di una frase, una buona conoscenza di alcuni principi linguistici della lingua di destinazionein questo caso è necessario, ad esempio l'ordine delle parole ecc . non sono richieste reali capacità di programmazione, ma la conoscenza di JAPE e Java di base sono necessari.

**Customizzazione del POS Tagger**

In questa parte ho voluto fare un resoconto passo passo di tutto il processo perché può risultare complicato soprattutto a quelli che sono alle prime armi con GATE e vogliamo fornire un’idea chiara di tutto il processo di ragionamentio e progettazione. Ci sono due strade per customizzare il POS Tagger di GATE:

La prima strada , utilizza il plugin **Tagger\_Framework[[42]](#footnote-42)** . E’ possibile installarlo direttamente dalla interfaccia grafica di GATE,al suo interno interno troviamo le seguenti risorse :

**C:\Program Files\GATE\_Developer\_7.1\plugins\Tagger\_Framework\resources\TreeTagger**

****

**Fig.58:Contenuto della cartella Tree Tagger**

Essa contiene già dei file .gapp per il parser dei documenti di varie lingue Francese, Tedesco, Spagnolo e anche una versione Italiana, *ma fornisce solo il file .gapp per l’implementazione e il test , non fornisce il parser della lingua italiana*; notare che manca il file *tree-tagger-italian* o *tree-tagger-italian-gate* il suffisso gate del secondo caso sta a indicare che quello è un parser perfettamente integrato con GATE, con l’italiano non saremo cosi fortunati.

A questo punto dando un’ occhiata ai file .gapp noteremo che abbiamo due opzioni per il file .gapp:



Entrambe le versioni funzionano, l’unica differenza tra la prima e la seconda, è che la prima fa uso di una particolare regola JAPE **sentencestring.jape**, che è possibile trovare alla seguente cartella:

**C:\Program Files\GATE\_Developer\_7.1\plugins\Tagger\_Framework\resources\tagger-independent\sentencestring.jape.**

Essa se richiamata dall’elemento Generic Tagger nel file .gapp, fa Tokenizzazzione e divisione delle frasi (cioè Tokenizer + Sentence Splitter) nello stesso momento, tenete conto che è una regola semplicissima che va a richiamare prima tutte le Sentence del documento che è la stesssa cosa di applicare il Sentence Splitter e poi va ad applicarci sopra la tokenizzazione di tipo Unicode (Unicode Tokenizer), consiglio di guardarla almeno una volta . Andiamo a vedere a livello di codice la differenza (che poi è l’unica importante) nel Generic Tagger:

|  |  |
| --- | --- |
| **<!—GENERIC TAGGER WITH NO TOKENIZATION-->**  **<gate.util.persistence.AnalyserRunningStrategyPersistence>**  **<runMode>1</runMode>**  **<featureName></featureName>**  **<featureValue></featureValue>**  **<pr class= "gate.util.persistence.PRPersistence" >**  **<runtimeParams class= "gate.util.persistence.MapPersistence" >**  **<mapType>gate.util.SimpleFeatureMapImpl</mapType>**  **<localMap>**  **<!—ALTRE ENTRY CON VALORI DI DEFAULT-->**  **<entry>**  **<string>taggerBinary</string>**  **<gate.util.persistence.PersistenceManager-URLHolder>**  **<urlString>$gatehome$custom/pos\_tagger\_IT/tree-tagger-italian-gate</urlString>**  **</gate.util.persistence.PersistenceManager-URLHolder>**  **</entry>**  **<entry>**  **<string>taggerDir</string>**  **<null/>**  **</entry>**  **</localMap>**  **</runtimeParams>**  **<resourceType>gate.taggerframework.GenericTagger</resourceType>**  **<resourceName>TreeTagger-IT-NoTokenization</resourceName>**  **<initParams class= "gate.util.persistence.MapPersistence" >**  **<mapType>gate.util.SimpleFeatureMapImpl</mapType>**  **<localMap>**  **<entry>**  **<string>preProcessURL</string>**  **<null/>**  **</entry>**  **<entry>**  **<string>postProcessURL</string>**  **<null/>**  **</entry>**  **</localMap>**  **</initParams>**  **<features class= "gate.util.persistence.MapPersistence" >**  **<mapType>gate.util.SimpleFeatureMapImpl</mapType>**  **<localMap/>**  **</features>**  **</pr>**  **</gate.util.persistence.AnalyserRunningStrategyPersistence>** | **<!—GENERIC TAGGER WITH TOKENIZATION-->**  **<gate.util.persistence.AnalyserRunningStrategyPersistence>**  **<runMode>1</runMode>**  **<featureName></featureName>**  **<featureValue></featureValue>**  **<pr class= "gate.util.persistence.PRPersistence" >**  **<runtimeParams class= "gate.util.persistence.MapPersistence" >**  **<mapType>gate.util.SimpleFeatureMapImpl</mapType>**  **<localMap>**  **<!—ALTRE ENTRY CON VALORI DI DEFAULT-->**  **<entry>**  **<string>taggerBinary</string>**  **<gate.util.persistence.PersistenceManager-URLHolder> <urlString>$gatehome$custom/pos\_tagger\_IT/tree-tagger-italian-gate</urlString>**  **</gate.util.persistence.PersistenceManager-URLHolder>**  **</entry>**  **<entry>**  **<string>taggerDir</string>**  **<null/>**  **</entry>**  **</localMap>**  **</runtimeParams>**  **<resourceType>gate.taggerframework.GenericTagger</resourceType>**  **<resourceName>TreeTagger-IT-Tokenization</resourceName>**  **<initParams class= "gate.util.persistence.MapPersistence" >**  **<mapType>gate.util.SimpleFeatureMapImpl</mapType>**  **<localMap>**  **<entry>**  **<string>preProcessURL</string>**  **<gate.util.persistence.PersistenceManager-UrlHolder>**  **<urlString>$gatehome$plugins/Tagger\_Framework/resources/**  **tagger-independent/sentencestring.jape</urlString>**  **</gate.util.persistence.PersistenceManager-UrlHolder>**  **</entry>**  **<entry>**  **<string>postProcessURL</string>**  **<null/>**  **</entry>**  **</localMap>**  **</initParams>**  **<features class= "gate.util.persistence.MapPersistence" >**  **<mapType>gate.util.SimpleFeatureMapImpl</mapType>**  **<localMap/>**  **</features>**  **</pr>**  **</gate.util.persistence.AnalyserRunningStrategyPersistence>** |

**Fig.59:Figura esplicativa per evidenziare la differenza nel Tagger con Tokenization(destra) e senza(sinistra)**

A questo punto dobbiamo creare il nostro file ***tree-tagger-italian-gate***, seguiamo la documentazione ufficiale di GATE e arriviamo alla pagina della documentazione del Tree Tagger[[43]](#footnote-43). Troviamo i file .par e i software, per la realizzazione e test dei propri file di parser, addirittura (si ringrazia *Ciarán Ó Duibhín*) vi è un’interfaccia grafica che rende possibile l’utilizzo anche da non esperti ,basta seguire la documentazione[[44]](#footnote-44).

*Lo scopo di questo studio non trovare un parser italiano perfetto per i documenti ,quindi ci limiteremo ad utilizzare le grammatiche italiane fornite dal file italian.par e italian-abbreviations.*

La documentazione fornisce diversi file .par per la lingua italiana noi utilizzeremo la versione fornita dall’Università, ma non cambia niente a livello concettuale. Una volta ottenuto il nostro file .par , seguendo le istruzioni della documentazione del software Tree Tagger ci ritroviamo con il file **italian.par** che fornisce la grammatica italiana e un file **italian-abbreviations** integra nella grammatica le abbreviazioni per esempio *sig.* invece di *signore* **.** Vediamo un possibile file script **tree-tagger-italian-gate :**

#!/bin/sh

############################################################

# SCRIPT FOR ITALIAN PART OF SPEECH TAGGING

###########################################################################

# THESE VARIABLES HAVE TO BE SET:

# For Windows, these should be Windows paths with backslashes (which will need single-quoting), e.g. BIN='C:\TreeTagger\bin'

# The script will probably fail if the BIN or LIB directory names contain spaces. You should install the tree tagger in a #directory that does not contain spaces in its path (i.e. not under "Program Files").

###############################################################################

#Il percorso varierà all’interno del progetto con riferimento alla gatehome e.g. $gatehome$/path/to/directory/of/file

BIN='C:\TreeTagger\bin'

CMD='C:\TreeTagger\cmd'

LIB='C:\TreeTagger\lib'

# set your perl compilator

#MWL=${CMD}/mwl-lookup.perl

# set your default options for the Tree Tagger

TAGGEROPTS="-token -lemma -sgml"

# set the path for the tagger command

TAGGER=${BIN}/tree-tagger

# set the path for the abbrevations file

ABBR\_LIST=${LIB}/italian-abbreviations

# set the path for the parameter file

PARFILE=${LIB}/italian.par

# remove empty lines

grep -v '^$' |

# recognition of MWLs

# -- disabled for use with GATE tokens

# $MWL -f ${LIB}/italian-mwls |

# tagging

$TAGGER $PARFILE -token

**Fig.60: Codice “rozzo” per il tree-tagger-italian-gate chi vuole può divertirsi a potenziarlo**

Per parsare i documenti italiani possiamo utilizzare il nostro POS Tagger italiano ci basta sostituire l’elemento POS Tagger di ANNIE con il nostro Tag Generico che fa riferimento a **tree-tagger-italian-gate.**

**Customizzazione del Gazetteer**

Customizzare il Gazetteer, è molto semplice, Gate lo permette direttamente dall’interfaccia, creiamo un nuovo modulo **ANNIE Gazzeter** e esattamente come per il modulo POS Tagger gli facciamo puntare l’urlString del gate.util.persistence.PersistenceManager-URLHolder alla risorsa voluta, il nostro file **myList.def** descritto precedentemente. Per richiamare le varie liste .lst dal nostro file .def esiste una struttura apposita:

**universita.lst:universita::it:Lookup**

**[nomeLista].lst : [majorType]:[minorType]:[lingua]:[\*\*]**

**\*\* = identifica l’elemento che “prende” le liste attraverso le nostre regole Jape , salvo particolari casi si utilizzerà l’input di default di GATE Lookup.**

**Di norma è lasciato vuoto (anche i “:” finali spariscono).**

Vediamo i parametri di Runtime <runTimeParams> del Gazetteer:

* **annotationSetName**: l’annotation set su cui vogliamo lavorare di default è “”;
* **longestMatchOnly**: se su una porzione di testo si sovvrapongono più annotazioni del Gazetter ad esempio “lab” e “laboratorio” prende solo quella più lunga quindi “laboratorio” di default è true;
* **wholeWordsOnly**: ogni parola del gazetteer deve corrispondere a una parola se true, cioè se la parola “lavora” è presente nel Gazetteer nel testo non voglio che annoti la parola”lavora” in “lavoratore” ma solo se vi è ache la parola “lavoratore” nel Gazetter;

Vediamo i parametri di init <initParams> del Gazetteer:

* **caseSensitive**: di default true;
* **encoding**: di default ASCII o UTF-8 a seconda della versione di GATE;
* **gazetteerFeatureSeparator**: questo è interessante , permette di settare per ogni singola parole del Gazetter una Feature assegnando un valore di separazione,per esempio prendiamo “&” come valore di separazione nella nostra lista abbiamo i valori “Scuola&Istituto=1” “Tribunale&Istituto=2” alle parole Scuola e Tribunale è stata assegnata la Feature Istituto con valori “1” e “2”. E’ quindi possibile richiamarla con regole JAPE con comandi come ({ LooukUp.Istituto}) dove prendera solo le parole del Gazetter con quela particolare Feature;
* **listsURL**: la path al nostro file .def e alle sue liste .lst;

*Ricordate che i rispettivi file .lst devono stare nella stessa directory (cioè nella stessa cartella/folder) dei loro file riassuntivi .def*

*La qualità del risultato aumenta se si utilizzano più file .def che identificano determinati insiemi di liste per esempio che tengano conto di liste costitutite da parole che iniziano con la maiuscola ,con la minuscola o con tutte e due caseUpperSensitive.def , caseLowerSensistive.def,caseUnsensitive.def; questo permette un utilizzo più mirato delle liste.*

**Customizzazione del main.jape di GATE o Semantic Tagger**

Per modificare i Semantic Tagger di GATE ovvero l’ ANNIE NE Transducer la cosa più semplice è andare a creare un nostro file main.jape (come descritto nell’APPENDICE C ) a cui si possono associare le regole Jape create attraverso le rispettive fasi.

Le regole JAPE nel nostro studio possono essere suddivise in tre categorie:

1. **Regole Jape di Preprocessing (le considereremo come le regole di supporto per le regole match)**: ne abbiamo create un paio, esse sono regole di inizializzazione per le successive regole JAPE ,sono regole molto blande che richiamano le varie liste e macro al fine di evitare problemi di collisione delle fasi o richiamo delle liste. Di solito il loro numero è consistente all’inizio della progettazione per verificare se utilizzare o meno certe liste e ridotte man mano che le regole Jape vengono migliorate e consolidate.
2. **Regole Jape di Taggatura degli elementi HTML:** Una delle cose principali da fare se non la più importante è identificare in quale parte del documento HTML sono state trovate le informazioni matchate (evidenziate ,catturate,..) dalle nostre annotazioni create con le regole Jape . Per esempio darò priorità alle informazioni contenuti nell’ HEAD o il FOOTER di un documento che a quelle contenute nel BODY .
3. **Regole Jape di Match delle Informazioni:** Se avete letto la veloce introduzione a Jape all’APPENDICE C, del documento allora avete un’idea di quante varianti per matchare delle informazioni in un testo si possono realizzare.

1. <https://gate.ac.uk/mail/> [↑](#footnote-ref-1)
2. Gerald Salton - Automatic Information Organization and Retrieval -McGraw-Hill Inc. 1968 [↑](#footnote-ref-2)
3. C. Manning, P. Raghavan, H. Schutze - Introduction to Information Retrieval”- Cambridge 2008 [↑](#footnote-ref-3)
4. http://it.wikipedia.org/wiki/ - Il data mining cerca patterns all'interno di grossi dataset ricavandoinformazione inizialmente sconosciuta; se fosse nota, sarebbe reperibile con opportune query [↑](#footnote-ref-4)
5. <http://en.wikipedia.org/wiki/Semantic_Web> [↑](#footnote-ref-5)
6. “profondo”, costituito da speciali database accessibili via web e siti dinamici, generalmente sconosciuti all'utente medio, la cui quantità di informazione supera la prima di 400-550 volte [↑](#footnote-ref-6)
7. <http://www.archivi.beniculturali.it/servizioII/progetti/ontologie.html> [↑](#footnote-ref-7)
8. *SIUSA* – *Sistema Informativo Unificato delle Sovrintendenze Archivistiche* e *Sistema Guida generale degli Archivi di Stato italiani* [↑](#footnote-ref-8)
9. Thomas R. Gruber - Translation Approach to Portable Ontology Specifications - appared in Knowledge Acquisition, 5(2):199-220, 1993 [↑](#footnote-ref-9)
10. <http://wordnetweb.princeton.edu> [↑](#footnote-ref-10)
11. <http://gate.ac.uk/> [↑](#footnote-ref-11)
12. H. Cunningham. A Definition and Short History of Language Engineering. - Journal of Natural Language Engineering, 5(1):1–16, 1999 [↑](#footnote-ref-12)
13. <http://gate.ac.uk/sale/tao/split.html> [↑](#footnote-ref-13)
14. [http://gate.ac.uk/sale/tao/splitch6.html#sec:misc-creole:reset](http://gate.ac.uk/sale/tao/splitch6.html%23sec:misc-creole:reset) [↑](#footnote-ref-14)
15. [http://gate.ac.uk/sale/tao/splitch6.html#sec:annie:tokeniser](http://gate.ac.uk/sale/tao/splitch6.html%23sec:annie:tokeniser) [↑](#footnote-ref-15)
16. [http://gate.ac.uk/sale/tao/splitch6.html#sec:annie:splitter](http://gate.ac.uk/sale/tao/splitch6.html%23sec:annie:splitter) [↑](#footnote-ref-16)
17. [http://gate.ac.uk/sale/tao/splitch6.html#sec:annie:regex-splitter](http://gate.ac.uk/sale/tao/splitch6.html%23sec:annie:regex-splitter) [↑](#footnote-ref-17)
18. [http://gate.ac.uk/sale/tao/splitch6.html#sec:annie:tagger](http://gate.ac.uk/sale/tao/splitch6.html%23sec:annie:tagger) [↑](#footnote-ref-18)
19. [http://gate.ac.uk/sale/tao/splitch21.html#sec:parsers:taggerframework](http://gate.ac.uk/sale/tao/splitch21.html%23sec:parsers:taggerframework) [↑](#footnote-ref-19)
20. <http://gate.ac.uk/sale/tao/splitap4.html#x34-712000D> [↑](#footnote-ref-20)
21. <http://tanl.di.unipi.it/it/pipe.html> [↑](#footnote-ref-21)
22. <http://gate.ac.uk/sale/tao/splitch13.html#sec:gazetteers:lkb-gazetteer> il capitolo 13.9.4 [↑](#footnote-ref-22)
23. [https://gate.ac.uk/gate/doc/plugins.html#JAPE\_Plus](https://gate.ac.uk/gate/doc/plugins.html%23JAPE_Plus) [↑](#footnote-ref-23)
24. [http://gate.ac.uk/sale/tao/splitch6.html#sec:annie:semantic-tagger](http://gate.ac.uk/sale/tao/splitch6.html%23sec:annie:semantic-tagger) [↑](#footnote-ref-24)
25. [http://gate.ac.uk/sale/tao/splitch6.html#sec:annie:orthomatcher](http://gate.ac.uk/sale/tao/splitch6.html%23sec:annie:orthomatcher) [↑](#footnote-ref-25)
26. <http://gate.ac.uk/sale/tao/splitch21.html#sec:misc-creole:ast> capitolo 21.14 [↑](#footnote-ref-26)
27. [https://gate.ac.uk/gate/doc/plugins.html#Tools](https://gate.ac.uk/gate/doc/plugins.html%23Tools) [↑](#footnote-ref-27)
28. <https://gate.ac.uk/sale/tao/splitch13.html#sec:gazetteers:flexgazetteer> [↑](#footnote-ref-28)
29. <http://gate.ac.uk/sale/talks/gate-course-aug10/track-3/module-10-advanced-ie/module-10-advanced-ie.pdf> [↑](#footnote-ref-29)
30. [http://gate.ac.uk/sale/tao/splitch15.html#sec:misc-creole:language-identification](http://gate.ac.uk/sale/tao/splitch15.html%23sec:misc-creole:language-identification) [↑](#footnote-ref-30)
31. [http://gate.ac.uk/sale/tao/splitch22.html#sec:misc-creole:lingpipe:langid](http://gate.ac.uk/sale/tao/splitch22.html%23sec:misc-creole:lingpipe:langid) [↑](#footnote-ref-31)
32. [http://gate.ac.uk/releases/gate-6.1-build3913-ALL/doc/tao/splitch3.html#x6-570003.7.2](http://gate.ac.uk/releases/gate-6.1-build3913-ALL/doc/tao/splitch3.html%23x6-570003.7.2) [↑](#footnote-ref-32)
33. <http://gate.ac.uk/family/embedded.html> [↑](#footnote-ref-33)
34. <http://gate.ac.uk/wiki/code-repository/src/sheffield/examples/StandAloneAnnie.java> [↑](#footnote-ref-34)
35. Un’esempio del codice che consiglio di guardare almeno una volta per avere una idea chiara su come programmare un proprio file .gapp/.xgapp, è possibile trovarlo all’indirizzo: <http://sourceforge.net/projects/optimacidoc/files/jape/RE_crm_eh/> . [↑](#footnote-ref-35)
36. <http://www.w3schools.com/googleAPI/default.asp> [↑](#footnote-ref-36)
37. <https://developers.google.com/maps/documentation/geocoding/?hl=it> [↑](#footnote-ref-37)
38. <http://www.json.org/java/> [↑](#footnote-ref-38)
39. <http://gate.ac.uk/sale/tao/splitch8.html#chap:jape> [↑](#footnote-ref-39)
40. <http://code.google.com/p/disko/source/browse/trunk/data/gate/plugins/ANNIE/resources/NE/?r=34> [↑](#footnote-ref-40)
41. <http://gate.ac.uk/sale/tao/splitap6.html> [↑](#footnote-ref-41)
42. <http://gate.ac.uk/sale/tao/splitch21.html> capitolo 21.3 [↑](#footnote-ref-42)
43. <http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/> [↑](#footnote-ref-43)
44. <http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html> [↑](#footnote-ref-44)