Università degli Studi di Bologna
Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

Laurea triennale

Named Entity Extraction: la piattaforma Gate/Annie

Relatore:
Prof. Dario Maio

Candidato:
Luca Campana

Anno Accademico 2014/2015

Indice

Indice delle figure

Glossario degli acronimi

ANNIE	A Nearly New Information Extraction system
COR	Coreference Resolution
CREOLE	Collection of REusable Objects for LE
EE	Event Extraction
GATE	a General Architecture for Text Extraction
IDE	Integrated Development Environment
IE	Information Extraction
IR	Information Retreival
JAPE	Java Annotation Pattern Engine
LE	Language Engineering
LHS	Left Hand Side
LP	Language Processing
LR	Language Resource
MT	Machine Translation
MUC	Message Understanding Conference
NE	Named Entity
NER	Named Entity Extraction
NLP	Natural Language Processing
POS	Part Of Speech
PR	Processing Resource
RE	Relationship Extraction
RHS	Right Hand Side

Abstract

In questo lavoro si introducono i concetti di base di *Natural Language Processing*, soffermandosi su *Information Extraction* e analizzandone gli ambiti applicativi, le attività principali e la differenza rispetto a *Information Retrieval*. Successivamente si analizza il processo di *Named Entity Recognition*, focalizzando l'attenzione sulle principali problematiche di annotazione di testi e sui metodi per la valutazione della qualità dell'estrazione di entità. Infine si fornisce una panoramica della piattaforma software *open-source* di *language processing* GATE/ANNIE, descrivendone l'architettura e i suoi componenti principali, con approfondimenti sugli strumenti che GATE offre per l'approccio *rule-based* a *Named Entity Recognition*.

1. Introduzione

Negli ultimi due decenni si è assistito ad un vertiginoso aumento della quantità dei contenuti digitali disponibili e scambiati, grazie soprattutto all'esplosione di servizi fruibili via Web e più in generale al processo di digitalizzazione che ha investito ogni ambito della comunicazione e della conoscenza umana. Tali meccanismi hanno permesso da un lato la riduzione dei costi nella produzione delle informazioni e dall'altro la globalizzazione di queste ultime, accessibili in modo semplice e rapido ad un numero sempre maggiore di persone in ogni lato del mondo.

Considerando il vastissimo ambito dei contenuti digitali prodotti in linguaggio naturale, è sorta, parallelamente alla diffusione della digitalizzazione delle informazioni, l'esigenza di munirsi di strumenti in grado di analizzare, comprendere e processare in modo automatico questi contenuti, con lo scopo di fornire servizi in grado di assistere le persone in un gran numero di attività, quali la traduzione di testi, l'estrapolazione di informazioni utili, l'attività di *question/answering*, la comprensione del linguaggio parlato ecc. É nata così una branca dell'intelligenza artificiale definita *Natural Language Processing* (NLP), che si occupa proprio dello sviluppo di strumenti per il trattamento di informazioni in linguaggio naturale. Lo scopo principale è quello di fornire alle macchine una capacità di comprensione e di produzione del linguaggio naturale simile a quella dell'essere umano. É possibile suddividere NLP in due attività principali:

- *natural language understanding*: analisi e comprensione del linguaggio naturale;

- *natural language generation*: produzione di informazioni in linguaggio naturale a partire da una data rappresentazione.

Considerando l'attività di *natural language understanding* è possibile fare un'ulteriore distinzione in *speech understanding* e *language understanding*: con *speech understading* si intende la comprensione del linguaggio parlato mentre con *language understanding* si intende la comprensione di testi scritti.

NLP è suddiviso in una moltitudine di *sub-task*, ognuno con il suo specifico raggio d'azione: uno dei più interessanti è *Information Extraction* (IE), cioè il processo di estrazione di informazioni strutturate da documenti non strutturati o semi-strutturati tipicamente in linguaggio naturale. Con informazioni non strutturate o semi-strutturate intendiamo dati il cui contenuto non è organizzato secondo un modello preciso o lo è comunque solo parzialmente; tale mancanza di strutturazione rende inefficiente il trattamento di questi dati da parte delle macchine. Uno degli scopi di IE è proprio quello di mettere in condizione tali sistemi di processare dati non strutturati e di permettere la produzione di inferenze a partire da quelli strutturati.

In questo lavoro si fornisce preliminarmente una panoramica di NLP e IE nell'ambito di *language understanding* nel capitolo 2, per poi approfondire un *sub-task* di IE, *Named Entity Recognition* (NER), nel capitolo 3 e una piattaforma tra le più importanti per *language processing*, GATE, nel capitolo 4.

2. Natural Language Processing e Information Extraction

2.1 Natural Language Processing

NLP è un campo dell'intelligenza artificiale il cui compito è analizzare ed elaborare i linguaggi naturali, cioè i sistemi di comunicazione comunemente usati dagli esseri umani in forma scritta, orale o gestuale. La ricerca in questo ambito si è focalizzata in particolare sui meccanismi che permettono alle persone di comprendere il contenuto di una comunicazione umana e sullo sviluppo di *tool* che possano fornire ai sistemi informatici la capacità di comprendere ed elaborare il linguaggio naturale. Le ricerche su NLP hanno dunque necessariamente riguardato diversi ambiti della conoscenza umana, come le scienze informatiche, l'intelligenza artificiale, la linguistica e la psicologia. Più nel specifico, la ricerca su NLP ha il compito di fornire ai computer gli strumenti per:

- assistere gli essere umani in compiti che riguardano la comprensione e l'elaborazione di documenti, anche multimediali, in linguaggio naturale (traduzioni da lingua a lingua, gestione di documenti ecc.);
- comunicare con gli esseri umani (assistenti vocali);
- analizzare, estrarre, ridurre o classificare informazioni di documenti scritti in linguaggio naturale *(information extraction*, indicizzazione per motori di ricerca, *summarization* ecc.);
- estendere le proprie conoscenze linguistiche, tramite tecniche di apprendimento automatico.

2.1.1 Cenni storici

Le prime ricerche su NLP iniziarono negli anni ’40 nel campo dei processi di *machine translation* (MT). Nei primi lavori in questo ambito si utilizzavano liste di *lookup* per tradurre le parole e regole per queste ultime secondo il sistema sintattico della lingua target una volta tradotte; tale approccio non teneva conto delle possibili ambiguità delle parole e delle strutture sintattiche, e i risultati ottenuti furono scarsi (Liddy, 2001).

Nella seconda metà degli anni ’60, l’organizzazione ALPAC (*Automatic Language Processing Advisory Committee*) stilò un documento in cui si sconsigliava di finanziare ulteriori ricerche su MT, a causa dell’elevato costo e delle scarse prestazioni dei sistemi dell’epoca (Hutchins, 1986); questo report ebbe un grande impatto negli Stati Uniti e in Europa, rallentando di fatto il progresso dell’intero settore di NLP.

Negli anni ’80, grazie alla maggiore disponibilità di risorse e alla introduzione di algoritmi di *machine learning*, le ricerche ricominciarono progressivamente.

Dagli anni ’90 in poi il settore del NLP è stato oggetto di un numero crescente di studi, grazie alla enorme disponibilità di testi scritti in formato elettronico, alle maggiori capacità dei calcolatori, alla diffusione di Internet e del Web e al conseguente forte interessamento di grandi compagnie del campo dell'IT (Liddy, 2001). Attualmente le ricerche si stanno concentrando su algoritmi di *machine learning* di tipo *unsupervised* o *semi-supervised*, in grado di derivare regole da documenti parzialmente o non annotati.

2.1.2 Passi di Natural Language Processing

NLP è una attività piuttosto complessa a causa della intrinseca ambiguità dei linguaggi umani e della necessità di possedere un'ampia conoscenza pregressa per poter comprendere a fondo il contenuto di gran parte delle comunicazioni umane; tuttavia nella maggior parte dei casi non è richiesto ad applicativi di NLP la totale comprensione di una comunicazione, ma solo di parti salienti di questa.

Ad alto livello, possiamo pensare il processo NLP come suddiviso in diversi livelli di analisi, corrispondenti, in parte, a quelli dell'analisi del linguaggio (Dale, 2010).

- Analisi lessicale: processo di suddivisione di un flusso di caratteri in porzioni atomiche definite *token*.
- Analisi sintattica: arrangiamento dei *token* estratti in strutture sintattiche, come ad esempio alberi sintattici (*parse tree*).
- Analisi semantica: assegnazione di un significato alla struttura sintattica, se necessario tramite disambiguazione.
- Pragmatica: processo di assegnazione di un significato alla struttura sintattica sulla base del contesto e di informazioni pregresse.

Spostandosi più a basso livello, i principali passi di analisi ed elaborazione del testo alla base di NLP sono qui di seguito elencati.

1. *Tokenization*: divisione di un flusso di caratteri in *token* (una parola, un simbolo di punteggiatura, un numero). Solitamente nei linguaggi naturali questo processo consiste nella separazione dei caratteri in corrispondenza di spazi o simboli di punteggiatura, ma tale informazione non è sempre sufficiente o disponibile.
2. *Stemming*: consiste nell'estrarre la radice dalla forma flessa di

una parola. La radice non corrisponde necessariamente al lemma (detto anche "radice morfologica", la quale rappresenta tutte le forme di una flessione per convenzione) della parola.

3. *Lemmanization*: processo che estrae il lemma a partire dalla forma flessa di una parola. Tale operazione può risultare complessa in quanto per effettuare una corretta disambiguazione è necessario comprendere il contesto in cui la parola è utilizzata.
4. *Finding collocation*: una *collocation* è un'espressione formata da due o più parole il cui significato non può essere dedotto da quello delle sue singole parti.
5. *Word sense disambiguation*: consiste nel precisare il significato di una parola (o di un insieme di parole), che presenta significati diversi a seconda del contesto in cui è usata.
6. *Anaphora resolution*: è il processo di ricerca delle relazioni tra porzioni di testo distanti all'interno di un documento. Tali relazioni sono definite come "rapporti anaforici".
7. *Part of speech* (POS) *tagging*: processo di associazione tra una parola di un testo il suo ruolo all'interno del discorso. POS determina dunque se una parola è un articolo, un nome, un aggettivo, una forma verbale ecc.

2.1.3 Approcci a Natural Language Processing

Esistono storicamente due tipi di approcci al problema di NLP, riportati appresso.

1. *Rule Based*: consiste in un elevato numero di regole definite manualmente. Il vantaggio principale di tale approccio è il forte controllo che si ha nella definizione delle regole di *language processing* (LP); tuttavia richiede elevate capacità e una grande

quantità di lavoro da parte degli sviluppatori.

2. *Machine Learning*: si tratta di tecniche di apprendimento automatico della macchina. Tipicamente si fornisce alla macchina un esteso *corpora,* cioè un insieme di documenti appartenenti al dominio di interesse annotati manualmente, per definire le corrette informazioni che devono essere apprese dal calcolatore. Tale soluzione risulta spesso meno dispendiosa rispetto all'approccio *rule based* e più facilmente adattabile a differenti domini applicativi ma richiede un elevato numero di documenti da sottoporre inizialmente alla macchina.

2.1.4 Principali tool

Di seguito sono elencate le più diffuse architetture per NLP.

- GATE/ANNIE: software gratuito e open-source nato all'interno della *University of Sheffield*, è una suite di strumenti per LP. ANNIE è il set di algoritmi specializzati in IE su testi non strutturati. L'architettura di GATE è analizzata approfonditamente nel capitolo 4 di questo lavoro.
- Stanford CoreNLP: è un *framework open-source* per l'analisi e l'elaborazione di testi, sviluppato alla *Stanford University* e scritto in linguaggio Java.
- UIMA: sviluppato da *Apache Software Foundation*, è un software per l'analisi di testi, video e audio non strutturati, scritto in Java e C++.
- OpenNLP: sviluppato da *Apache Software Foundation*, è un software per LP scritto in Java. Utilizza algoritmi *machine learning*.

2.2 Information Extraction

IE è un *subtask* di NLP e rappresenta il processo di identificazione in un testo di predefiniti insiemi di concetti appartenenti ad uno specifico dominio (Jakub Piskorski, 2012). Lo scopo di IE è, più precisamente, l'estrazione di contenuto semantico rilevante strutturato (appartenente ad un preciso e circoscritto contesto) da documenti non strutturati o semi-strutturati. Per fare un esempio concreto si consideri innanzitutto il documento con tema "*joint venture*" utilizzato durante la conferenza *Message Understanding Conference*-5 (MUC-5), svoltasi nel luglio 1993 (Hobbs, Fastus, 1997), riportato qui appresso.

> *Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan. The joint venture, Bridgestone Sports Taiwan Co., capitalized at 20 million new Taiwan dollars, will start production in January 1990 with production of 20,000 iron and 'metal wood' clubs a month.*

Le conferenze MUC erano competizioni tra sistemi di IE sviluppati da diverse organizzazioni. Facendo riferimento al testo citato poco sopra, uno dei compiti dei sistemi di IE partecipanti a MUC-5 era quello di estrarre automaticamente dal testo dato i dettagli della "*joint venture*", come le entità coinvolte (in questo caso "*Bridgestone Sports Co.", "a local concern*" e "*a Japanese trading house*"), la capitalizzazione della nuova società ("NT$20000000") e il nome della nuova società mista ("*Bridgestone Sports Taiwan Co*."); il processo

di IE permette allora di associare un contenuto semantico a una porzione di testo e di generare una nuova informazione strutturata a partire dalle singole entità estratte.

Lo scopo di IE è dunque quello di estrarre le sole informazioni utili tralasciando quelle superflue, affinché possano poi essere consultabili rapidamente da un essere umano e manipolabili efficacemente da una macchina; non è uno scopo primario di IE invece la completa comprensione del significato di un testo, comunque impossibile da raggiungere attualmente a causa di limiti tecnici (Jakub Piskorski, 2012).

2.2.1 Task dell'Information Extraction

IE comprende diversi sub-*task*, di cui si fornisce appresso una breve descrizione.

- *Named Entity Recognition* (NER): riconoscimento e classificazione di entità presenti nel testo all'interno di specifiche categorie, come nomi propri (di persone, di luoghi ecc.), espressioni di tempo, percentuali ecc. Ad esempio, considerando il testo "Stanley Kubrick è nato a New York il 26 luglio 1928. È considerato uno dei maggiori cineasti della storia del cinema.", un tipico risultato di un processo di NER può essere la categorizzazione di "Stanley Kubrick" come "Persona", "New York" come "Luogo" e "26 luglio 1928" come "Data".
- *Coreference Resolution* (COR): ricerca di espressioni di coreferenza, cioè dei rinvii a una medesima entità all'interno di un testo. Esistono diversi modi per esprimere una coreferenza, tra cui pronominale, nominale e implicito. Considerando il testo

"I Police registrarono l'album Synchronicity nel 1983. Quello fu il loro ultimo album.", un processo di COR è in grado di associare l'entità "album" al pronome dimostrativo "Quello", indicando che si sta ancora facendo riferimento al medesimo album.

- *Relationship Extraction* (RE): riconoscimento e classificazione dei legami tra le entità. Applicando tale processo alla frase "Michelangelo Buonarroti nacque a Caprese il 6 marzo 1475." e considerando l'esistenza di una relazione di tipo "*nato_a*", otterremo un risultato del tipo *nato_a*("Michelangelo Buonarroti ", "Caprese").
- *Event Extraction* (EE): identificazione e strutturazione di eventi. Ad esempio, il processo di EE sul testo "Facebook ha acquisito Whatsapp nel 2014, ad un prezzo di 19 miliardi di dollari." è in grado di identificare l'acquirente ("Facebook"), la società acquisita ("Whatsapp"), l'anno in cui è avvenuta la vendita ("2014") e il costo dell'operazione ("14 miliardi di dollari").

2.2.2 Differenza tra Information Extraction e Information Retrieval

Molto spesso i concetti di IE e *Information Retrieval* (IR) sono tra loro confusi, dunque è importante fare una chiara distinzione tra le due attività.

Il compito di IR è di selezionare da una collezione di documenti testuali un sottoinsieme di documenti pertinenti rispetto ad una *query* (Jakub Piskorski, 2012), cioè quel set di documenti dove è probabile

sia presente l'informazione cercata. Solitamente i documenti sono associati ad una misura di rilevanza, calcolata in svariati modi: spetterà poi all'utente cercare ed estrarre dai documenti restituiti le informazioni cercate.

IE, a differenza dei processi di IR, non si occupa semplicemente di restituire un sottoinsieme di documenti ma, basandosi sull'analisi del linguaggio naturale, è in grado di estrarre informazione utile, strutturata e non ambigua da testi non strutturati o semi-strutturati. Tali informazioni potranno poi essere inserite in un database, in un foglio di calcolo o essere usati come indici per attività di IR.

2.2.3 Diffusione di Information Extraction

Negli ultimi anni la ricerca su IE ha fatto notevoli progressi e i sistemi di IE sono oggigiorno sempre più utilizzati come strumento automatico di reperimento dati per analisi finanziarie, per attività di intelligence, per ricerche di mercato, in ambito biomedico, medico, chimico, per monitorare trend sui social network ecc. Due tra gli svariati fattori che hanno permesso una più rapida diffusione di IE sono:

- il vertiginoso aumento della quantità di documenti digitali disponibili, la cui gestione è divenuto un problema di primaria importanza; per fare un esempio, solo nell'ambito medico vengono scritti più di mezzo milione di articoli all'anno (Hobbs, Information Extraction, 2010). Oggigiorno la problematica principale non è l'esistenza di una data informazione (si presuppone che l'informazione ricercata sia presente e raggiungibile) ma la sua ricerca ed estrazione, in un contesto in cui la maggior parte delle informazioni risultano non strutturate o

semi-strutturate. La vastità dei testi disponibili ha fatto dunque nascere l'esigenza di possedere strumenti atti all'estrapolazione automatica di informazioni strutturate da questi;

- l'organizzazione di progetti come MUC, finanziato a partire dagli anni ’80 dal governo degli Stati Uniti d'America, da cui emersero buona parte delle tecnologie di IE e i metodi per la misurazione della qualità e dell’attendibilità dei risultati.

La ricerca su IE si è concentrata tradizionalmente sull’estrazione di informazioni da documenti in lingua inglese; negli ultimi anni però, dato l’aumento del numero di documenti disponibili anche in altre lingue, sono stati sviluppati sistemi di IE per documenti non in lingua inglese (Jakub Piskorski, 2012). Le ricerche in questo settore cresceranno probabilmente nei prossimi anni, favorendo la diffusione di IE.

3. Named Entity Recognition

NER è un *subtask* di IE, specializzato nella ricerca e classificazione di *Named Entity* (NE), cioè di porzioni di testo di documenti in linguaggio naturale che rappresentano entità del mondo reale (Leon Derczynski, 2014), come nomi di persone, luoghi, dati, compagnie ecc. Il termine NE è stato utilizzato per la prima volta all'evento MUC-6 (David Nadeau, 2006), durante il quale si comprese l'importanza di riconoscere nomi propri ed espressioni numeriche all'interno di un testo per svolgere un corretto processo di IE. NER è un subtask fondamentale di IE in quanto:

- pone le basi per una analisi più approfondita del testo, eseguita ad esempio tramite i processi di COR, RE e EE utilizzando le NE estratte;
- effettua una scrematura estraendo solo le entità utili in un determinato contesto.

Oggi molte grandi compagnie di Information Technology fanno uso di NER, ad esempio per riconoscere e categorizzare entità in pagine web o per analizzare in modo automatico il contenuto di documenti. Altri ambiti in cui NER è stato utilizzato con successo sono la bioinformatica, la chimica e la medicina. In Figura 1: attività di Named Entity Recgnition eseguita con GATE è mostrato un processo di NER eseguito su di una semplice frase utilizzando il software GATE.

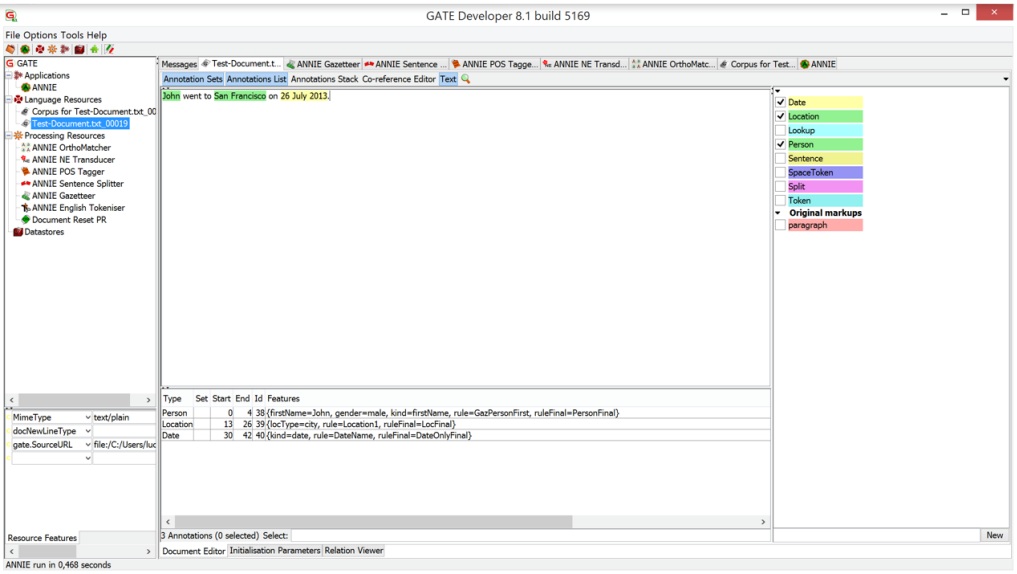

Figura 1: attività di Named Entity Recgnition eseguita con GATE.

3.1 Passi di NER e approcci utilizzati

Il processo di NER è in linea generale suddiviso in due passi:

- ricerca dei nomi e segmentazione, cioè individuazione dei confini;
- classificazione, assegnando a specifiche porzioni di testo annotazioni con validità semantica.

Esistono tre tipi di approcci per individuare e la classificare le entità:

- tramite *gazetteer* e *lookup list*, cioè utilizzando liste di parole suddivise per categorie e preferibilmente attinenti al contesto applicativo. Ha lo svantaggio di richiedere liste estremamente lunghe, spesso non sufficienti per contesti applicativi complessi;
- tramite *pattern*, definiti all'interno di un insieme di regole, che

se rispettati determinano la categoria dell'entità. É definito anche come approccio *rule-based*. Tali regole analizzano il contesto e le annotazioni già apposte all'entità per determinarne la categoria. La stesura di regole richiede programmatori esperti e una lunga fase di *testing* successiva.

- tramite tecniche di *machine learning*. Partendo da un corpus il sistema è in grado di derivare alcuni modelli di tipo probabilistico basati sulla sequenza delle parole, utilizzati poi per derivare annotazioni. L'approccio tramite *machine learning* richiede estesi corpora di *training*.

Il *tool* di GATE per IE, ANNIE, fa ad esempio un uso combinato di *lookup list* e regole.

Considerando la classificazione, non esiste ad oggi uno standard condiviso: negli anni sono emerse diverse interpretazioni riguardo all'organizzazione e all'estensione dell'insieme delle categorie. Uno degli standard proposti, emerso durante una delle conferenze MUC, prevede l'annotazione delle entità tramite formato XML e le seguenti categorie base.

- Valore numerico, contrassegnato dall'etichetta "NUMEX", per
  - soldi;
  - percentuali.
- Tempo, contrassegnato dall'etichetta "TIMEX", per
  - date;
  - ore.
- Nome proprio, contrassegnato dall'etichetta "ENAMEX", per
  - persone;

- organizzazioni;
- luoghi.

I tipi di entità coinvolte in un processo di NER e la loro granularità dipendono comunque dal dominio applicativo a cui si sta facendo riferimento. Ad esempio, facendo riferimento all'entità di tipo "luogo", è possibile definire sotto-categorie più precise come "città", "stato" o "nazione" (David Nadeau, 2006).

3.2 Valutazione della qualità

La valutazione di un sistema di NER è basata tradizionalmente sul confronto tra il risultato del processo di NER effettuato su un corpus dal sistema automatico e il risultato della annotazione sullo stesso corpus da parte di annotatori umani. Le principali misure di valutazione della qualità di un software di NER sono le seguenti:

- precisione (P): percentuale di entità corrette tra tutte quelle estratte;
- tasso di errore: è l'inverso della precisione, dunque rappresenta la percentuale di entità errate tra tutte quelle estratte;
- recupero (R): percentuale di entità corrette estratte tra tutte quelle correte presenti;
- F-score (F): rappresenta la media armonica pesata tra precisione e recupero. Essendoci infatti un *trade-off* tra precisione e richiamo, si è reso necessaria l'introduzione di una misura di calcolo delle performance che tenesse conto di entrambi gli aspetti. F-score è definito dalla seguente formula:

$$F = \frac{(\beta^2 + 1)\mathrm{PR}}{\beta^2 P + R}$$

dove β definisce il peso che si vuole attribuire a precisione e recupero. Se β = 1, hanno la medesima importanza, se β > 1 la precisione ha una maggiore importanza, se β < 1 il recupero ha più importanza.

Attualmente i più avanzati sistemi NER possiedono un'accuratezza simile a quella di un essere umano per testi in inglese, ma solo se concepiti specificamente per un dato ristretto dominio e se utilizzati su corpora di piccole dimensioni. Ad esempio durante la conferenza MUC-7 il punteggio F-score più alto ottenuto da un sistema automatico è stato di 93.39, mentre annotatori umani hanno ottenuto un punteggio compreso tra 97.60 e 96.95 (Named Entity Scores - English, 2001).

3.3 Problematiche e sfide future

L'attività di NER presenta diverse problematiche legate soprattutto all'ambiguità e alla complessità del linguaggio umano.

Alcune criticità nell'estrazione di NE sono dovute a:

- ambiguità del linguaggio: per esempio si ha ambiguità lessicale quando una medesima parola può in contesti differenti avere significati diversi;
- variabilità di NE: ad esempio l'entità persona "Mario Bianchi" potrebbe trovarsi espressa in diverse forme: "Signor Bianchi", "Bianchi", "Mario" ecc.;

- metonimie, cioè quei meccanismi di “trasferimento di significato da una parola a un’altra in base a una relazione di contiguità spaziale, temporale o causale” (Treccani, 2015). Ad esempio l'entità Inghilterra può essere categorizzata come “organizzazione” o come “località”, a seconda del contesto. Nella frase “L'Inghilterra ha vinto la Coppa del Mondo”, Inghilterra si riferisce alla Nazionale Inglese di calcio, e dovrebbe essere categorizzata dunque come “organizzazione”; al contrario, nella frase “La Coppa del Mondo si è tenuta in Inghilterra” l'entità Inghilterra fa riferimento ad una “località”.

Attualmente, nonostante gli ottimi risultati ottenuti dai sistemi di NER, le sfide principali sono la riduzione del lavoro di annotazione dei corpora, grazie all'utilizzo di tecniche di *semi-supervised machine learning*, l'aumento delle performance nell'utilizzo di NER su ampi e diversificati domini e una maggiore specificità e precisione nella definizione delle entità.

4. GATE

GATE (*General Architecture for Text Engineering*) è un sistema *open-source* gratuito sviluppato dall'Università di Sheffield che offre agli utenti una piattaforma completa di LP. Più nello specifico, GATE è contemporaneamente un'architettura, un *Integrated Development Environment* (IDE), un *Framework* e una *Web App* (Cunningham, 2014).

- Come architettura, GATE definisce un'organizzazione ad alto livello dei sistemi per LP e assicura una corretta interazione tra i componenti.
- Come IDE (GATE *Developer*), aiuta gli utenti a sviluppare in modo semplice ed efficiente applicazioni per NLP, fornendo una GUI e funzionalità per il *debugging.*
- Come *Framework (*GATE *Embedded),* offre una libreria di componenti che può essere inclusa dagli sviluppatori in vari applicativi ed eventualmente estesa.
- Come *Web App* (GATE *Teamware*), permette di annotare collettivamente un set di documenti.

GATE è dunque un sistema completo di *Language Engineering* (LE), che include componenti per LP come *parsers*, strumenti di *machine learning*, *stemmers*, strumenti per la manipolazione di testi e l'inserimento di annotazioni, *tool* per IE e per effettuare valutazioni e benchmark e un linguaggio per la scrittura di regole. Il suo sviluppo è iniziato nel 1995 e attualmente il suo team di sviluppo è il più grande nell'ambito del software *open-source* indirizzato a NLP (Cunningham, 2014).

4.1 GATE IDE

Gate IDE fornisce un'interfaccia grafica per lo sviluppo di applicazioni di NLP, e più nello specifico permette in modo intuitivo di annotare ed editare documenti, creare corpora, utilizzare le risorse fornite dal *framework* di GATE.

Facendo riferimento alla Figura 2 possiamo suddividere l'interfaccia di GATE in diverse parti:

- nella parte superiore ci sono i menù *Files*, *Option*, *Tools* e *Help* e i bottoni relativi alle azioni più comuni;
- nella colonna di sinistra, in alto, appaiono le risorse caricate nella sessione corrente;
- nella colonna di sinistra, in basso, sono visualizzate informazioni riguardo alle risorse caricate;
- al centro abbiamo il riquadro principale, che offre il dettaglio sulle risorse selezionate;

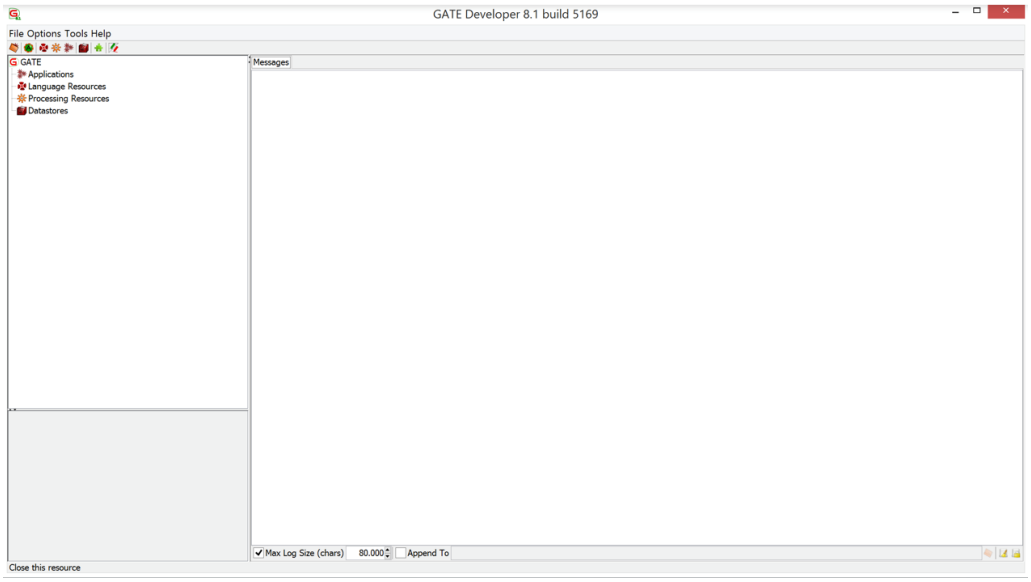

Figura 2: interfaccia grafica di GATE IDE.

4.1.1 Caricamento di documenti e corpora

GATE supporta il caricamento di molti formati di documenti, sia locali che online; è possibile infatti caricare un documento sia tramite *browsing* dei file locali contenuti nel proprio *hard disk* sia immettendo un URL di una pagina web. Il documento importato sarà rappresentato all'interno di GATE come *Language Resource* (LR). É inoltre presente un editor di documenti, tramite cui è possibile modificare un documento o visualizzarne le proprietà e le annotazioni.

Utilizzando i documenti caricati è possibile creare corpora; i documenti possono essere aggiunti a un corpus anche dopo che questo è stato creato. Selezionando un corpus viene visualizzata la lista dei documenti che lo compongono: tramite l'editor è possibile rimuovere o aggiungere documenti al corpus e definire i parametri iniziali di quest'ultimo.

4.1.2 Annotazioni

Uno dei compiti principali di GATE è l'annotazione di documenti: tale operazione può essere fatta manualmente, semi-manualmente, aggiungendo annotazioni a documenti già precedentemente processati e annotati o automaticamente, utilizzando i componenti forniti da GATE. È possibile definire nuovi tipi di annotazioni o modificare quelle già esistenti.

Tramite l'interfaccia grafica è possibile gestire e analizzare le annotazioni presenti nel documento: ad esempio GATE permette di selezionare le tipologie di annotazioni visibili sul testo e per ogni annotazione permette di visualizzarne le proprietà e i dettagli. Inoltre grazie alla modalità di visualizzazione "*Annotation Stack"* (Figura 3) è possibile analizzare in modo chiaro le annotazioni sovrapposte, cioè quelle annotazioni di tipologie diverse relative alla medesima porzione di testo.

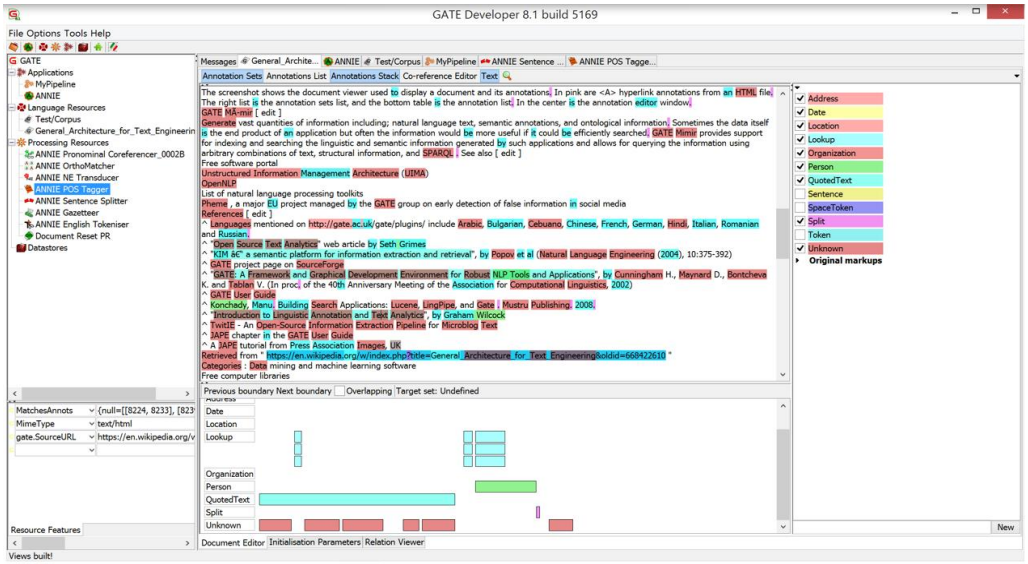

Figura 3: modalità di visualizzazione *Annotation Stack* di GATE.

4.1.3 Coreferenza

GATE Developer offre la possibilità di estrarre o editare catene di coreferenza, cioè set di *token* riferiti alla medesima entità, grazie ad una specifica sezione definita *Co-reference Editor* (Figura 4). Ogni catena di coreferenza prenderà in automatico il nome dell'elemento con il maggior numero di caratteri presente all'interno della stessa. Gli elementi inseriti all'interno di una catena di coreferenza possono essere rimossi da questa. GATE offre inoltre la possibilità di creare catene di coreferenza manualmente, a partire dalle annotazioni presenti nel documento.

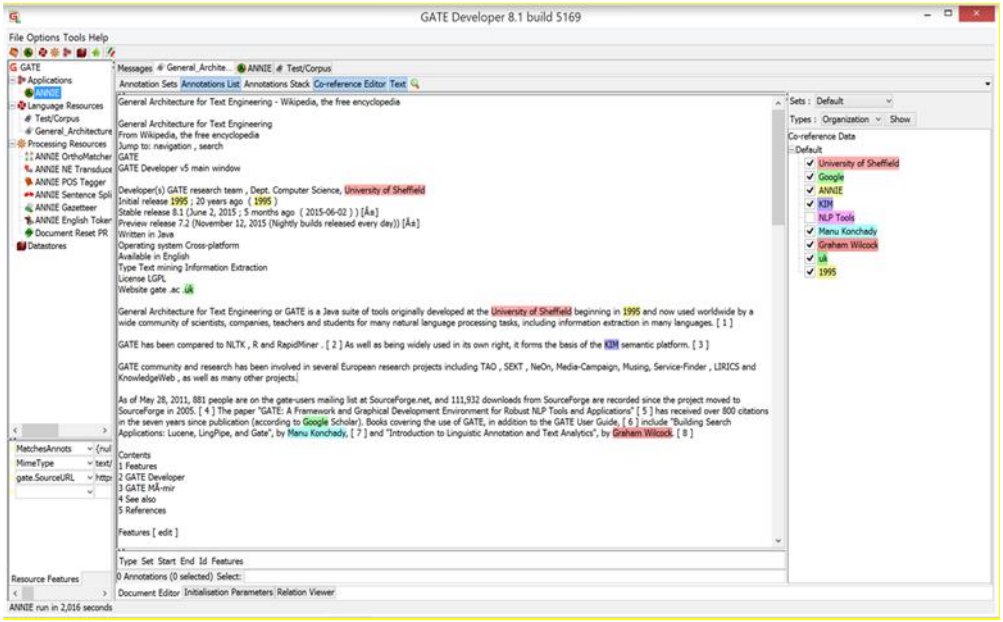

Figura 4: *Co-reference Editor* di GATE.

4.1.4 Caricamento di Processing Resources

Processing resources (PR) sono un set di componenti messi a disposizione da GATE, ognuno con funzionalità specifiche nell'ambito di LP. Sono trattate con maggior dettaglio nel paragrafo 4.3.

Per poter utilizzare PR su un corpus, dopo averle caricate ed aver eventualmente modificato i parametri iniziali, è necessario inserirle all'interno di a*pplication* (applicazioni), cioè controller che ne regolano il flusso di esecuzione. Creata una nuova applicazione, si scelgono i componenti che dovranno essere eseguiti, l'ordine di esecuzione, i documenti o il corpus che devono essere processati ed eventuali condizioni per l'esecuzione di PR; a questo punto è possibile lanciare l'applicazione. GATE offre di default applicazioni che comprendono già uno specifico set di PR e che possono essere utilizzate rapidamente senza particolari configurazioni: una di queste è ANNIE, i cui componenti sono trattati in modo approfondito nel paragrafo 4.4.

4.1.5 Persistenza delle risorse

Esistono tre modalità di salvataggio dei documenti annotati in GATE:

- utilizzando il formato originale del documento. Al documento potranno essere aggiunte solo le annotazioni ritenute utili, escludendo tutte le altre.
- utilizzando il formato proprietario di GATE, derivato da XML, in grado di memorizzare tutte le annotazioni;
- scrivendo un proprio algoritmo per persistere il documento.

Per corpus di documenti piuttosto estesi, che possono saturare la memoria se aperti tutti contemporaneamente, GATE offre la possibilità di salvataggio su *datastore*: in questo modo i documenti verranno salvati su disco e aperti solo quando necessario.

Infine in GATE è possibile persistere le applicazioni create e il loro stato: in questo modo un'applicazione potrà essere caricata da

file anche in un secondo momento. Da notare che questo tipo di salvataggio su file comprende la persistenza dello stato dell'applicazione ma non delle risorse utilizzate: ciò significa che caricando l'applicazione su una macchina diversa da quella su cui è avvenuto il salvataggio possono emergere errori legati all'assenza di tutte le risorse necessarie e alla posizione di queste nel *file system*, che potrebbe non coincidere con quella indicata sul file (che corrisponde alla posizione delle risorse nella macchina in cui il file è stato generato). Nel caso in cui si desideri salvare un'applicazione per usarla su più macchine, per garantire il corretto funzionamento delle risorse, è possibile esportarla utilizzando la modalità "*Export for GATE-Cloud.net*". GATE crea infatti in questa modalità un file ZIP contenente non solo lo stato dell'applicazione ma anche tutte le risorse utilizzate: in questo modo il pacchetto può essere utilizzato su più macchine, in quanto contiene già al suo interno tutto il materiale necessario per il funzionamento dell'applicazione.

4.2 JAPE

JAPE (*Java Annotation Patterns Engine*) è un linguaggio di programmazione di *pattern-matching* per GATE derivato da CPSL (*Common Pattern Specification Language*), che permette di scrivere e applicare regole ad annotazioni presenti nel testo: queste regole vengono infatti tradotte in codice Java e applicate sulle annotazioni del testo in input, producendo in output nuove annotazioni o la modifica di quelle già esistenti.

Un *JAPE grammar* è un set di fasi di elaborazione disposte secondo un preciso ordine e in cui ogni fase è un set di regole e azioni; tali fasi sono eseguite in modo sequenziale creando così una cascata di trasduttori a stati finiti, ognuno dei quali prende come input l'output (e le relative annotazioni) della fase precedente. L'ordine di esecuzione delle fasi è definito nel file *main.jape*. Una fase è strutturata come riportato qui di seguito.

- Intestazione che contiene gli attributi:
  - Phase: il nome della fase;
  - Input: il tipo di dati in input su cui vengono eseguite le operazioni*;*
  - Options: opzioni di controllo delle regole.
- Definizione delle macro da utilizzare (opzionale);
- Definizione di *template* da utilizzare (opzionale). Un *template* è una variabile a cui è associato un valore;
- Set di regole JAPE.

Qui di seguito è riportato un esempio di fase, con una sola regola.

```
Phase: JobTitle
Input: Lookup
Options: control = appelt debug = false


Rule: Jobtitle1
Priority: 80
(
 {Lookup.majorType == jobtitle}
(
  {Lookup.minorType == jobtitle}
)?
```

```
):jobtitle
-->
:jobtitle.JobTitle = {rule = "JobTitle1"}
```

In questo caso la fase si aspetta come input almeno un *Lookup*, mentre la sezione *Options* indica che il metodo per il *matching* è *appelt* e che la fase non è eseguita in modalità *debug*. É inoltre definita una regola, chiamata *JobTitle1*, che possiamo suddividere in diverse parti, come descritto appresso.

- Nome della regola.
- Priorità (opzionale), indica una priorità tra le regole che agiscono sulla medesima porzione di testo ed è espressa come numero intero positivo; un valore più alto indica una maggiore priorità. Se non specificato il valore della priorità è impostato di default a -1.
- LHS (*Left Hand Side*), che consiste nella descrizione del pattern per cui la regola deve trovare il match. È la parte di codice compresa tra la priorità e il simbolo "→"; ogni pattern definito deve essere racchiuso tra parentesi graffe.
- RHS (*Right Hand Side*), descrive le azioni da eseguire sull'annotazione nel caso in cui questa rispetti il pattern definito da LHS. È la parte di codice che segue il simbolo "→".

In conclusione, un JAPE *grammar* è costituito da un insieme di fasi eseguite in sequenza su un documento, in cui ogni fase è composta da un set di regole, che si contendono l'input all'interno della stessa fase.

4.2.1 Modalità di esecuzione delle regole

Ogni fase può avere cinque differenti valori dell'attributo *control*, qui di seguito elencati, che determina le modalità di esecuzione delle regole.

- *brill*: se una o più regole individuano un match nella medesima porzione di documento, vengono eseguite tutte contemporaneamente. É possibile avere quindi più annotazioni sulla medesima parte di testo. Tutte le regole vengono eseguite a partire da una stessa posizione e il processo di *matching* successivo partirà dalla posizione in cui il match più lungo termina.
- *all:* è simile a *brill*, ma la regola prosegue la ricerca di match dall'offset seguente rispetto a quello corrente.
- *first*: ogni regola è eseguita appena viene riscontrato un match, senza cercare il match più lungo.
- *once*: appena è individuato un match e lanciata una regola, l'esecuzione della fase termina.
- *appelt*: solo una regola può essere eseguita sulla stessa porzione di testo, secondo determinate regole: di tutte le regole che riscontrano un match a partire dalla stessa posizione nel testo, verrà lanciata quella che individua la più lunga regione di match. Se più di una regola trova un match per la stessa regione viene lanciata quella con più alta priorità. Se due o più regole hanno la stessa priorità viene lanciata quella definita prima delle altre.

Per fare un esempio di scenario in cui due regole trovano match sulla stessa porzione di testo, con la proprietà *control* in modalità *appelt*, si faccia riferimento al codice qui di seguito e al testo campione

"*China sea*" già in precedenza annotato, con la parola "China" annotata dal *gazetteer* come "*Location*" e "*sea*" definito come "loc_key" (nome di località) di tipo "post".

```
Rule: Location1
Priority:25
(
({Lookup.majorType == loc_key,Lookup.minorType == pre}
{SpaceToken})?
Lookup.majorType == location}
({SpaceToken}
{Lookup.majorType == loc_key, Lookup.minorType == post})?
)
:locName -->
:locName.Location = {kind = "location", rule = "Location1"}

Rule: GazLocation
Priority: 20
(
({Lookup.majorTyp == location}):location
)
→ :location.Name = {kind = "location", rule=GazLocation}
```

In questo caso, entrambe le regole trovano match (i meccanismi con che permettono di trovare match all'interno di un testo verranno descritti nei sottoparagrafi 4.2.2 e 4.2.3), ma solo la prima delle due è eseguita, perché partono entrambe dallo stesso punto e in quanto la prima regola possiede una regione di match più estesa rispetto alla seconda regola. Si consideri ora invece come testo di esempio la sola parola "*China*"; anche in questo scenario entrambe le regole potrebbero essere lanciate, ma la priorità di "Location1" è più alta e dunque ha la precedenza.

4.2.2 Left Hand Side

LHS è quella parte di *JAPE grammar* che definisce il pattern da rispettare affinché una porzione annotata di testo possa essere processata secondo le regole definite da RHS della regola stessa. Le annotazioni in input e le loro proprietà verranno quindi confrontate con il pattern espresso in questa parte della regola. La definizione di un pattern è racchiusa tra parentesi graffe e può essere fatta in diversi modi, come appresso descritto:

- specificando una stringa di testo (ad esempio Token.string == "is");
- specificando una tipologia di annotazione assegnata dai moduli precedenti (ad esempio "Lookup");
- specificando proprietà e valore di una annotazione (ad esempio "Token.kind == number");
- specificando meta-proprietà di una annotazione (a cui si accede tramite il simbolo "@"), come la sua lunghezza o la stringa a cui fa riferimento (ad esempio "X@length > 5");

JAPE supporta inoltre molti operatori, elencati qui di seguito:

- uguaglianza ("==") e disuguaglianza ("!=");
- operatori di comparazione (">", ">=", "<", "<=");
- operatori di *regular expression* ("=~", "==~", "!~", "!=~");
- *contexual operators* ("*contains*", "*notContains*", "*within*", "*notWithin*") per controllare se una annotazione comprende o è compresa in un'altra.

4.2.3 Costruzione di pattern complessi

GATE offre la possibilità di combinare le definizioni di più pattern,

in modo da definire strutture più complesse. Ad esempio più pattern possono essere combinati in un'unica regola e cercare matching in sequenza, come nell'esempio seguente.

```
Rule: InLocation
(
 {Token.category == "IN"}
 {Location}
):inLoc
```

Nell'esempio la regola individua un match con un *token* di categoria "IN", seguito da un'annotazione di tipo "*Location*". Quando si utilizzano pattern in sequenza è importante ricordare che il corretto riconoscimento di un match dipende dai valori definiti nell'attributo *Input* della fase a cui la regola appartiene. Possiamo immaginare infatti che la stringa annotata come "IN" e la stringa con annotazione di categoria "*Location*" saranno probabilmente separate da uno spazio: la regola descritta sopra individuerà un match solo se non sono stati dichiarati come valori dell'attributo *Input* tipi di annotazioni che possano essere presenti tra i due pattern. Se ad esempio fosse dichiarata l'annotazione "*SpaceToken*", che rappresenta uno spazio, molto probabilmente non verrebbe individuato alcun match.

È possibile associare i pattern tra loro racchiudendoli tra parentesi tonde, come nell'esempio seguente.

```
Rule: InLocation
({Token.category == "IN"} | ({Token.category == "JJ"})
     {Location}
):inLoc
```

JAPE supporta anche i seguenti operatori di ripetizione:

- “?”, se l'individuazione del match con il pattern è opzionale;
- “*”, se sono necessari 0 o più match;
- “+”, se sono necessari 1 o più match.

É possibile inoltre definire un numero preciso di match di un pattern, racchiudendo un numero intero positivo tra parentesi quadre (ad esempio [3]), o un *range* di match (ad esempio [2,5]).

In una stessa regola JAPE è possibile avere più di un pattern e più di un’azione; ogni pattern dovrà essere racchiuso tra parentesi tonde e avere una propria *label* univoca all'interno della regola ed ogni azione dovrà essere associata ad una *label*, come nell’esempio qui di seguito.

```
Rule: PersonJobTitle
Priority: 20

(
{Lookup.majorType == jobtitle}
):jobtitle
(
{TempPerson}
):person
-->
:jobtitle.JobTitle = {rule = "PersonJobTitle"},
:person.Person = {kind = "personName", rule = "PersonJobTitle"}
```

4.2.4 Templates

JAPE permette di definire cosiddetti *templates*, utili nel caso in cui un valore o una stringa siano frequentemente utilizzati all'interno di un *JAPE grammar*: un *template* è un nome associato ad un valore, a

cui può essere fatto riferimento in ogni parte di un *grammar* semplicemente riportando il nome con cui il *template* è stato definito. I *template* vengono definiti come di seguito:

```
Template: myString= "This is my string"
Template: threshold = 0.6
```

Come si può facilmente intuire il grande vantaggio di definire template è il risparmio di tempo e la probabilità minore di errori nel caso in cui sia necessario cambiare tutte le occorrenze di un valore molto utilizzato in un grammar. Definendo un template sarà sufficiente cambiare il valore espresso nella dichiarazione di questo per propagare il cambiamento a tutte le occorrenze. I template possono inoltre contenere parametri ed essere annidati tra loro.

4.2.5 Macro

Le macro sono pattern identificati da una stringa, utili per evitare la definizione ripetuta del medesimo pattern all'interno di una regola. É possibile utilizzare le macro all'interno delle regole riportando la stringa identificativa racchiusa tra parentesi tonde. L'esempio di macro riportato qui di seguito serve ad esempio per identificare *token* che esprimono i concetti di “milione” e “miliardo”.

```
Macro: MILLION_BILLION
(
{Token.string == "m"}|
{Token.string == "million"}|
{Token.string == "b"}|
{Token.string == "billion"}|
{Token.string == "bn"}|
{Token.string == "k"}|
```

```
{Token.string == "K"}
)
```

4.2.6 Right Hand Side

RHS è la parte di regola che contiene le istruzioni per creare o modificare un'annotazione. Un'annotazione, come spiegato più approfonditamente nel paragrafo 4.3, appartiene ad una categoria ed è caratterizzata da un set di attributi (*feature*). La porzione di testo su cui è applicabile una nuova annotazione o su cui è possibile effettuare modifiche di una annotazione già definita in precedenza è contraddistinta da una etichetta temporanea, associata solo nel caso in cui quel frammento di documento abbia trovato match con un pattern riferito alla medesima etichetta. Ad esempio la regola:

```
Rule: GazLocation
(
{Lookup.majorType == location}
)
:location -->
:location.Enamex = {kind="location", rule=GazLocation}
```

esprime che, se c'è match tra il valore dell'attributo "*majorType*" dell'annotazione "*Lookup"* e il pattern definito in LHS, allora alla parte di testo interessata dal match è applicata una *label* temporanea, in questo caso definita come "*location*". A questo punto RHS annota la parte di testo con label "*location*", aggiungendo a questa un'annotazione di tipo "*Enamex*" con attributo "*kind*" di valore "*location*" e un attributo "*rule*" con valore "*GazLocation*", che segnala la regola con cui è stata inserita questa annotazione.

Possono essere utilizzate macro per definire parti codice di RHS;

è necessario includere la *label* nella macro, che dovrà ovviamente corrispondere a quella di LHS. Di seguito lo schema esemplificativo della struttura di una macro per RHS.

```
Macro: MY_MACRO
: label
{
     // action
}
Rule: MyRule
(
  ({Token.string == "Something"})
:match) -->  MY_MACRO // no label here, only macro name
```

4.2.7 Copia di valori da Left Hand Side a Right Hand Side

JAPE permette il riferimento di valori degli attributi delle annotazioni anche in RHS, secondo le modalità mostrate nell'esempio qui di seguito.

```
Rule: LocationType
(
{Lookup.majorType == location}
):loc
-->
:loc.Location =
{rule = "LocationType", type = :loc.Lookup.minorType}
```

Nella definizione del valore dell'attributo "*type*" della nuova annotazione "*Location*", si fa riferimento al valore della dell'attributo "*minorType*" appartenente all'annotazione già esistente "*Lookup*". Nel

caso in cui la *feature* a cui si fa riferimento non esista, il nuovo attributo non apparirà nell'annotazione. In RHS si può fare riferimento, oltre alle proprietà, anche alle meta-proprietà di un'annotazione, tramite il simbolo "@".

4.2.8 Codice JAVA in Right Hand Side

RHS può essere definito anche in codice Java. Un esempio di RHS con Java è mostrato qui di seguito.

```
Rule:FirstName

(
{Lookup.majorType == person_first}
):person
-->
{
AnnotationSet person = bindings.get("person");
Annotation personAnn = person.iterator().next();
FeatureMap features = Factory.newFeatureMap();
features.put("gender", personAnn.getFeatures().get("mi-
norType"));
features.put("rule", "FirstName");
outputAS.add(person.firstNode(), person.lastNode(), "FirstPer-
son",
features);
}
```

Questa regola ha il compito di inserire una proprietà di tipo "*gender*" al nome proprio di una persona, basandosi sul valore di "*minorType*", già annotato dai componenti precedenti. Per prima cosa si ottiene il set di annotazioni della porzione di testo annotata temporaneamente come "*person"*. Una volta ricavata dal set l'annotazione cercata si crea una nuova collezione di attributi (*FeatureMap*) e si aggiunge a

tale collezione l'attributo "*gender"*, il cui valore è quello ricavato dall'attributo "*minorType"*. Si aggiunge inoltre una proprietà "*rule"* che identifica il nome della regola che è stata applicata. Infine si crea una nuova annotazione chiamata "*FirstPerson*", a cui aggiungiamo gli attributi appena definiti e che ha i medesimi punti di inizio e fine sul documento dell'annotazione "*person"*.

4.3 GATE Embedded

GATE Embedded è un *framework object-oriented* implementato in Java ed è il vero e proprio cuore dell'architettura GATE. Può essere utilizzato dagli sviluppatori per inserire funzionalità di NLP all'interno dei propri applicativi. GATE Embedded è un software *open-source*, con licenza *GNU Lesser General Public License 3.0*.

Possiamo immaginare Gate Embedded come un'interfaccia a cui possono essere connessi vari componenti; questi componenti forniscono le funzionalità base per attività di LP. GATE *Embedded* è suddiviso in diverse API, come mostrato in Figura 5, che vengono analizzate nei successivi sottoparagrafi.

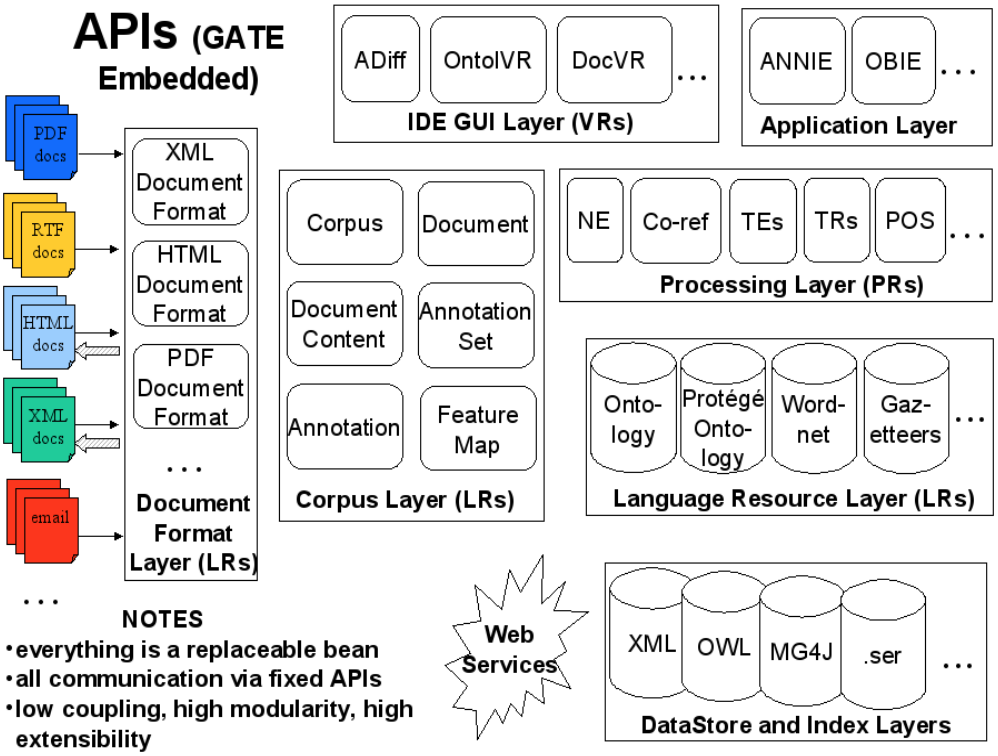

Figura 5: architettura di GATE Embedded.

4.3.1 CREOLE

L'architettura di GATE è fortemente basata su componenti, cioè su porzioni di software indipendenti tra loro che implementano ristrette funzionalità per i vari ambiti di LP: l'insieme di tali componenti è definito CREOLE (*a Collection of REusable Objects for Language Engineering*). I componenti di GATE, definiti anche come *resources*, sono di tre tipi.

- LanguageResources (LR): sono entità come il lessico, i corpora, le annotazioni, le ontologie.
- ProcessingResources (PR): sono gli algoritmi di LP, come *parsers*, *POS-taggers* ecc.
- VisualResources (VR): sono le risorse che generano la GUI.

Tale distinzione evidenzia l'importanza, nell'architettura di GATE, della separazione dei compiti tra i vari componenti, per ragioni di manutenibilità, estendibilità e di facilità di utilizzo.
I componenti più importanti forniti da GATE sono:

- LR per la gestione di documenti, corpora e annotazioni;
- PR per la definizione degli algoritmi di LP;
- *gazzetters*;
- *ontologie;*
- *tool* per *machine learning*;
- *parser* e *tagger*.

Gli elementi che costituiscono CREOLE possono essere facilmente caricati in modo indipendente proprio grazie al *framework* fornito da GATE, che costituisce un'interfaccia (o *backplane*) su cui effettuare il "*plug*" dei componenti. L'utente fornisce al sistema un elenco di URL e il sistema si occuperà di caricare i componenti localizzati negli indirizzi specificati. L'unione del *framework* (*backplane*) e dei componenti rappresenta la porzione di software che può essere esportata negli applicativi sviluppati dagli sviluppatori.

Tutte le risorse di CREOLE sono associate ad un file XML "*creole.xml*", che definisce le proprietà, i parametri necessari e i valori di default dei componenti: quando una risorsa deve essere caricata, viene ricercato il file *creole.xml* relativo all'URL della risorsa fornito dall'utente e il suo contenuto è utilizzato per inizializzarne i parametri e per caricare le classi necessarie, solitamente contenute all'interno di file *jar*. Inoltre, tutte le risorse di GATE sono basate su classi *Java Bean*, il modello di Java per componenti software. Il modello *Java Bean* è utilizzato per incapsulare più oggetti all'interno di

un unico oggetto detto *bean*, che garantisce una maggiore gestibilità. *Java Bean* impone il rispetto di determinate convenzioni alle classi che vogliono aderire a tale modello, con lo scopo di aumentare la riusabilità e la facilità di sostituzione dei *bean*.

4.3.2 Language Resources

LR rappresenta quelle tipologie di componenti che gestiscono i documenti, i corpora e le annotazioni. Le proprietà di ogni LR sono gestite tramite collezioni di coppie chiave/valore, definite come *FeatureMap*.

- Documenti: i documenti in GATE sono rappresentati come l'unione di contenuto, annotazioni e meta-data. Il contenuto di un documento deve implementare l'interfaccia "*gate.DocumentContent*", le annotazioni sono invece inserite all'interno di set definiti come *AnnotationSet*, mentre i meta-dati sono inseriti all'interno di *FeatureMap*.
- Corpora: un corpus è modellato in GATE come una lista di documenti.
- Annotazioni: le annotazioni sono meta-dati associati a una particolare sezione di un documento. Le annotazioni sono organizzate in GATE come grafi aciclici (Figura 6), in cui i nodi rappresentano specifiche parti del contenuto di un documento (puntatori ad una specifica posizione del documento rappresentata da un offset) mentre gli archi rappresentano le annotazioni. Un'annotazione è definita da un ID, un tipo, un nodo di partenza, uno di destinazione, e diversi attributi, inseriti all'interno di *FeatureMap*.

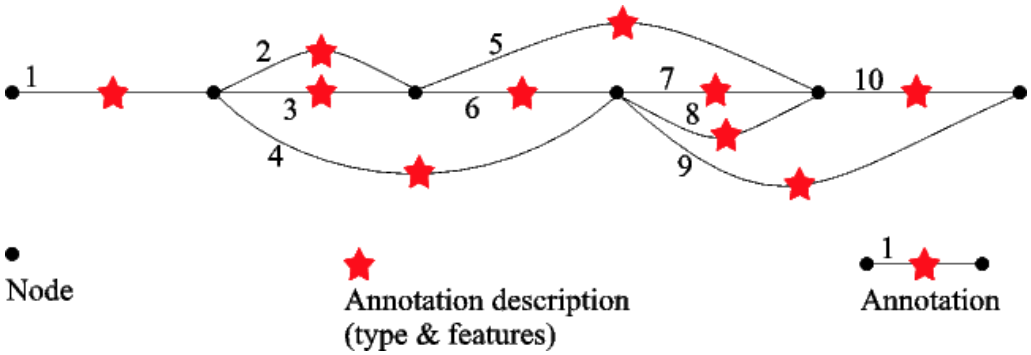

Figura 6: organizzazione delle annotazioni in GATE.

In Figura 7 viene mostrato il risultato di tre processi di annotazione su un singolo periodo: *tokenization* con *POS-tagging*, "*name recognition*" e "*sentence boundary recognition*".

Text

```
Cyndi savoredthe soup.
^0...^5...^10..^15..^20
```

Annotations

Id	Type	SpanStart	Span End	Features
1	token	0	5	pos=NP
2	token	6	13	pos=VBD
3	token	14	17	pos=DT
4	token	18	22	pos=NN
5	token	22	23	
6	name	0	5	name_type=person
7	sentence	0	23	

Figura 7: risultato del processo di annotazione su un frammento di testo.

GATE, oltre ai testi semplici, supporta diversi formati di tipo testuale, come XML, RTF, HTML, PDF, SGML, email, alcuni formati di Microsoft Office e alcuni formati di OpenOffice. La compatibilità con altri formati può essere estesa tramite plugin. Nel momento in cui viene importato un nuovo documento, GATE prova a identificare

il formato del documento per poi convertire ogni *markup* in un'annotazione, secondo la struttura sopra descritta. Prendiamo come il seguente *markup*.

```
<aTagName attrib1="value1" attrib2="value2" attrib3="value3">
A piece of text
</aTagName>
```

Il risultato della conversione in annotazione sarà:

```
annotation.type = "aTagName";
annotation.fm =
{
attrib1=value1;atrtrib2=value2;
attrib3=value3
};
annotation.start = startNode;
annotation.end = endNode;
```

Dunque il nome del *markup* è utilizzato per definire il tipo dell'annotazione, gli attributi vengono inseriti all'interno di *FeatureMap* come coppie chiave-valore mentre gli offset *markup* rispetto al documento andranno a costituire il nodo di partenza e di destinazione dell'annotazione.
Infine GATE offre tre modalità per la persistenza di LR:

- salvataggio su database;
- salvataggio su file tramite *Java serialization;*
- salvataggio su file con formato interno basato su XML.

Inoltre i documenti potranno essere esportati nuovamente nel loro

formato originale, scegliendo eventualmente di includere o meno annotazioni.

4.3.3 Processing Resources

Le risorse di tipo PR rappresentano i veri e proprio algoritmi per le operazioni di LP; sebbene in grado di funzionare in modo autonomo possono, come già spiegato in precedenza, essere aggregate in cosiddette applicazioni o *controller*, cioè in controlli che ne regolano la sequenza e il flusso di esecuzione. GATE supporta solo esecuzioni in *pipeline*, cioè sequenziali, di risorse CREOLE.

Esistono due tipi di pipeline, descritte qui di seguito.

- *Simple pipelines*, cioè set di PR eseguiti sequenzialmente.
- *Corpus pipelines*, cioè set di PR eseguiti su documenti e corpora. Un c*orpus pipeline* apre il primo documento del corpus, lo imposta come parametro di tutti i PR del set, ed esegue tutti i PR sul documento. Tale processo viene eseguito per tutti i documenti del corpus.

I *controller* sono essi stessi PR, e ciò permette di annidare una *pipeline* all'interno di un'altra. Sono inoltre presenti le versioni condizionali delle *pipeline*, che permettono di eseguire o meno un PR, sulla base di determinate condizioni decise dall'utente.

4.4 ANNIE

GATE offre una serie di componenti riusabili ed estendibili per l'esecuzione di task nell'ambito di NLP: come già ricordato, un sottoinsieme di questi componenti è stato accorpato per costituire ANNIE (*A Nearly New IE system*), un potente *tool* per IE basato su algoritmi

a stati finiti e su regole scritte in linguaggio JAPE. ANNIE è costituito da diversi componenti (Figura 8), disposti a formare una *pipeline* e qui di seguito elencati.

- Tokeniser.
- Gazzetter.
- Sentence splitter.
- Part Of Speech (POS) tagger.
- Semantic tagger.
- Orthographic reference.

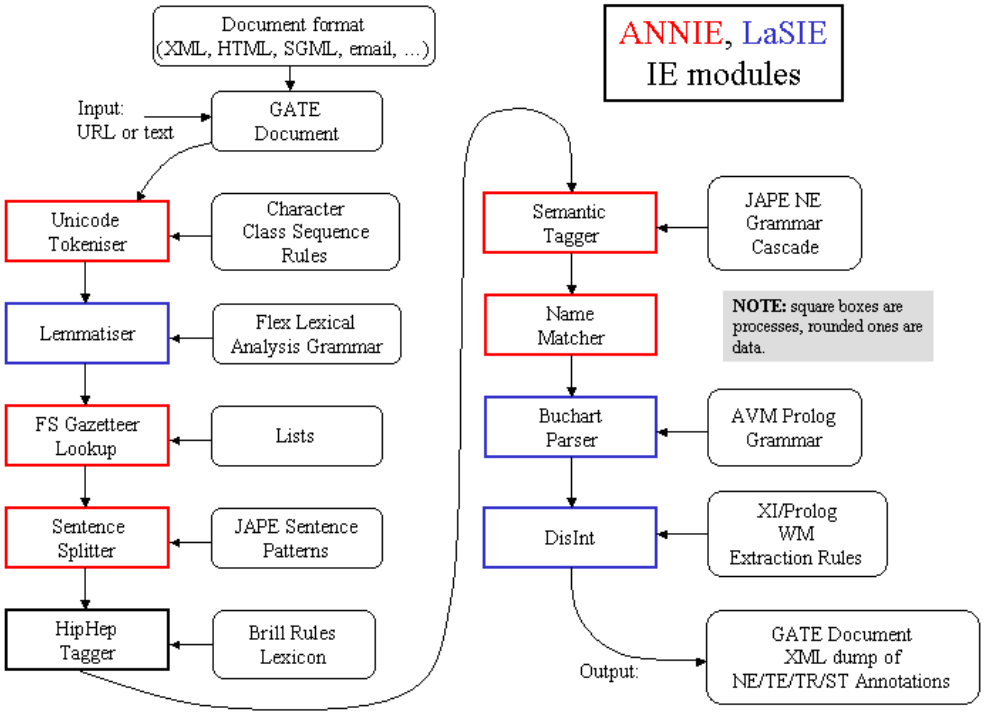

Figura 8: rappresentazione delle pipeline e dei componenti di ANNIE.

Nei seguenti sottoparagrafi si descrivono i più importanti componenti della *pipeline* di ANNIE più altre risorse non comprese in ANNIE ma caricabili separatamente come PR.

4.4.1 Document Reset

La risorsa *Document reset* permette di riportare il documento allo stato originario, rimuovendo tutte le annotazioni presenti. É possibile comunque decidere di resettare solo determinate annotazioni, lasciando inalterate le altre. È solitamente utilizzata all'inizio di una applicazione.

4.4.2 Tokeniser

Tokeniser ha il compito di frammentare il testo in componenti semplici, come numeri, punteggiatura, spazi, simboli, parole, che vengono definiti *token*. *Tokeniser* utilizza un approccio *rule-based*: qui di seguito è riportata la regola per identificare una parola che inizia con una sola lettera maiuscola:

```
"UPPERCASE_LETTER" "LOWERCASE_LETTER"* >
Token; orth=upperInitial; kind=word;
```

Se una porzione di testo rispetta questa regola viene annotata come *token*, di tipo "word" ("parola"). Esistono in GATE di default cinque tipologie di *token*.

1. *Word*: è definito come un qualunque set di caratteri maiuscoli o minuscoli (incluso il simbolo "-") contigui. Un *token* di tipo *word* possiede un attributo definito *orth*, che può assumere quattro valori:
   - upperInitial, se con prima lettera maiuscola e tutte le altre minuscole;
   - allCaps, se con tutte lettere maiuscole;
   - lowerCase, se con tutte lettere minuscole;

- mixedCase, se con insieme di lettere maiuscole e minuscole che non corrisponde alle precedenti categorie.

2. *Number*: è definito come un set di cifre contigue.
3. *Symbol*: può essere di due tipi, di valuta (ad esempio “€”, “$”) o standard (ad esempio “&”).
4. *Punctation*: sono definiti tre tipi di *punctation.*
   - start_punctation: ad esempio “(”;
   - end_punctation: ad esempio “)”;
   - other_ punctation: ad esempio “:”.
5. *Space Token*: rappresenta uno spazio bianco tra le parole e si suddivide in due categorie, *space* e *control*, a seconda che si tratti di un carattere di spazio o carattere di controllo.

Queste tipologie di *token* sono quelle definite nel *tokeniser* di default: è possibile, se necessario, creare altri *tokeniser* a seconda delle proprie esigenze.

In ANNIE è compresa un PR definito *English Tokeniser*, unione di un normale *tokenizer* e di un *JAPE transducer*. Il compito di *JAPE transducer* è quello di adattare l'output generico di un *tokenizer* ai requisiti richiesti da *POS-tagger* per la lingua inglese. Un compito di questo *transducer* è ad esempio quelli di processare il risultato di *tokenization* del costrutto “don't” per portarlo da tre token (“don”, “ ' ”, “t”) a due (“do”, “n't”).

4.4.3 Gazetteer

Il compito del *gazetteer* è quello di identificare le entità nel testo a partire da liste definite *lookup list*; ogni lista è un semplice file di testo contenente i nomi propri comuni relativi a diverse categorie di

entità, come nomi di persone, nomi di città, nomi di organizzazioni, giorni della settimana ecc.

Nella Figura 9 è mostrata una parte del contenuto del file city.lst, utilizzato dal *gazetteer* di GATE, contenente una lista di nomi di città.

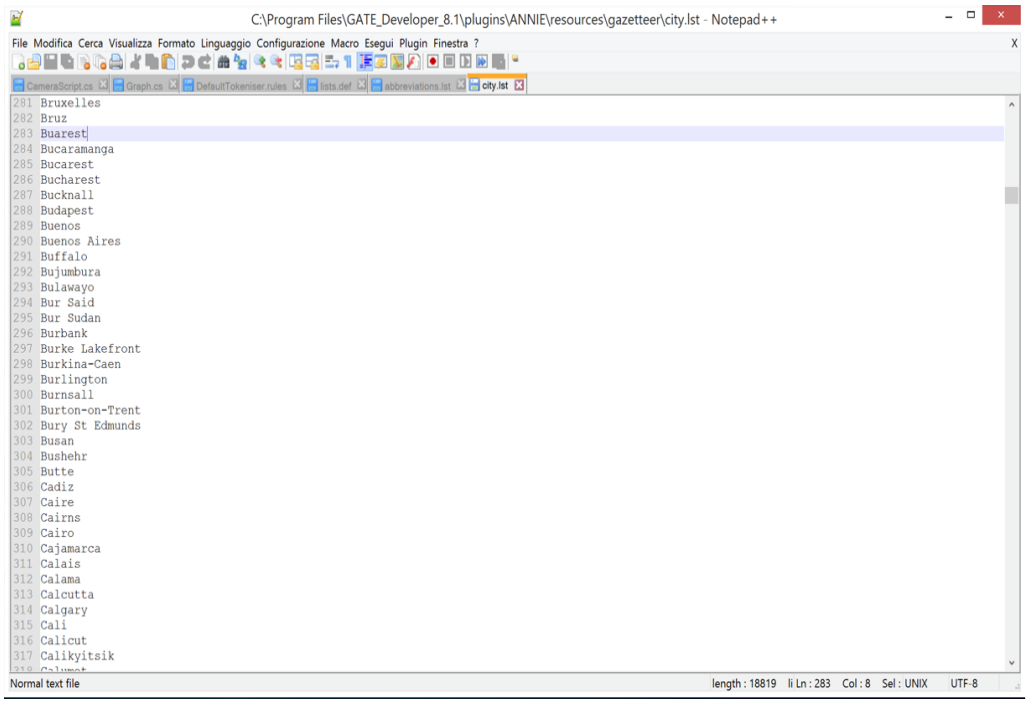

Figura 9: contenuto del file city.lst.

Per accedere alle liste il *gazetteer* utilizza un file indice index.def (Figura 10) in cui per ogni lista viene specificato un tipo principale ed eventualmente un tipo minore.

```
20 currency_prefix.lst:currency_unit:pre_amount
21 currency_unit.lst:currency_unit:post_amount
22 date_key.lst:date_key
```

Figura 10: contenuto del file *index.def*.

La prima colonna si riferisce al nome del file della lista, la seconda al tipo principale (es. "*currency_unit*"), la terza al tipo secondario (es. "*post_amount*"). Per ogni *token* del testo compatibile con una delle *entry* delle liste viene creata un'annotazione con i valori presenti nella colonna del tipo primario e del tipo secondario. Le *entry* delle liste possono essere modificate o eliminate, sia all'interno di GATE grazie al *Gazetteer editor* sia tramite un semplice editor di testo.

4.4.4 Sentence Splitter

Il *sentence splitter* ha il compito di suddividere un testo in periodi, cioè in porzioni di testo dotate di significato e organizzate secondo precise regole grammaticali e sintattiche. Ogni periodo viene annotato come "*Sentence*", e ogni segno di terminazione del periodo viene annotato come "*Split*". Un'annotazione di tipo *split* ha un attributo "*kind"* che può essere di due tipi, *internal* se il segno di terminazione corrisponde ad un numero di punti da uno a quattro, a un punto esclamativo o interrogativo o alla combinazione di questi ultimi, *external* se corrisponde ad un ritorno a capo.

In alternativa a *Sentence splitter* standard di ANNIE è possibile utilizzare *RegEx Sentence Splitter*, basato su *regular expression* scritte in Java, più robusto e con performance migliori su input irregolari.

4.4.5 Part-Of-Speech Tagger

POS *tagger* ha il compito di individuare ed annotare le parti del discorso, cioè parole o simboli di un testo con proprietà o funzionalità simili che possono quindi essere organizzate in classi. Ad esempio,

le parti del discorso nella lingua italiana si distinguono tradizionalmente in diverse classi: nome, aggettivo, articolo, pronome, verbo, avverbio, preposizione, congiunzione e interiezione. Il numero delle classi e la loro definizione dipende dalla lingua a cui si sta facendo riferimento.

POS *Tagger* di GATE fa uso di un lessico e di regole derivate dall'esercitazione su un vasto corpus di articoli del *Wall Street Journal*; sia il lessico che le regole possono essere comunque modificate se necessario. Esistono poi altri due lessici addizionali, uno per i testi composti da soli caratteri in maiuscolo (*lexicon_cap*), l'altro per testi composti da soli caratteri in minuscolo (*lexicon_lower*).

4.4.6 Semantic Tagger (ANNIE Named Entity Transducer)

Semantic Tagger (chiamato in GATE "ANNIE NE *Transducer*") ha il compito di elaborare le annotazioni apposte durante le fasi precedenti, per generare entità annotate, disambiguando tra i vari significati possibili tramite il confronto con pattern. Ad esempio, la parola "*May*" ha diversi significati nella lingua inglese: può rappresentare il mese di maggio ("*May 2013"*), un cognome ("*Brian May*") o un verbo ("*It may rain today"*). *Semantic Tagger* inoltre è in grado di combinare più annotazioni in singole entità: ad esempio nel caso di date espresse secondo il pattern giorno, mese, anno, viene creata un'unica annotazione di tipo "*Date*" che comprenderà tutte e tre le espressioni.

I tipi di annotazioni che possono di default essere assegnate in GATE sono basate sulla definizione delle entità di MUC:

- Person
  - gender: male, female

- Location:
  - locType: region, airport, city, country, county, province, other
- Organization:
  - orgType: company, department, government, newspaper, team, other
- Money
- Percent
- Date
  - kind: date, time, dateTime
- Address
  - kind: email, url, phone, postcode, complete, ip, other
- Identifier
- Unknown

Semantic Tagger di GATE è basato su regole scritte in linguaggio JAPE (Figura 11), che descrivono i pattern che devono essere rispettati. Parte delle annotazioni sono generate a partire dalle liste di *gazetteer*, dunque l'alterazione di queste ultime può portare ad una variazione dei risultati di questa fase. L'annotazione "*unknown*" viene utilizzata nella fase successiva dall'*OrthoMatcher* e rappresenta un

```
252 // IP Address Rules
253
254 Rule: IPaddress1
255 (   {Token.kind == number}
256     {Token.string == "."}
257     {Token.kind == number}
258     {Token.string == "."}
259     {Token.kind == number}
260     {Token.string == "."}
```

Figura 11: esempio di regola per trovare indirizzi IP nel testo, utilizzata da *Semantic Tagger* di GATE.

nome non ancora identificato.

4.4.7 Orthographic Coreference (OrthoMatcher)

Il compito principale di *Orthographic Coreference* è quello di individuare relazioni di identità tra le entità trovate dal *Semantic Tagger*, effettuando cioè una analisi delle coreferenze basandosi su informazioni di tipo ortografico. Ad esempio è possibile che in un testo si faccia più volte riferimento alla medesima entità, anche con abbreviazioni o acronimi. Le regole per definire una relazione di coreferenza vengono invocate tra due parole solo se queste sono già state categorizzate come entità dello stesso tipo o se una di loro è stata categorizzata come "*unknown*", in modo da non sovrascrivere le precedenti annotazioni. Per riconoscere relazioni di identità tra nomi che identificano la medesima entità ma che sono rappresentati ortograficamente in modo differente (ad esempio con pseudonimi o abbreviazioni) viene utilizzata una tabella di aliases (Figura 12).

```
 8 New York Times Inc.£4
 9 Times£4
10 New York Times£4
11 Coca-Cola Co.£5
12 Coca-Cola Co£5
13 Coca-Cola£5
14 Coke£5
```

Figura 12: contenuto della lista di alieses (del file alias.lst) utilizzata da *Orthographic Coreference* di GATE.

4.4.8 Pronominal Coreference

Il componente *Pronominal Coreference* ha il compito di risolvere le riprese anaforiche presenti nel testo utilizzando una serie di regole scritte in linguaggio JAPE ed è suddiviso in tre sottomoduli, descritti

appresso.

1. *Quoted text module:* ha il compito di identificare le porzioni di testo quotate, cioè comprese tra virgolette, per permettere la risoluzione dei pronomi presenti al loro interno. È basato su regole JAPE in grado di rilevare le parti di testo quotate e di annotarle come "*Quoted Text*": tali regole identificano nel testo simboli che possano indicare la presenza di un segmento di testo quotato.
2. *Pleonastic it module*: utilizza regole JAPE per individuare una ripresa pleonastica del pronome "*it*", cioè quel caso in cui "*it*" è usato nella medesima frase in cui è presente l'entità a cui è riferito.
3. *Pronominal resolution* module: ha l'obiettivo di effettuare risoluzioni pronominali, tramite i seguenti passi.
    - Preprocessamento del documento, per individuare annotazioni che serviranno negli step successivi e preparare le strutture dati.
    - Ricerca dei pronomi e per ognuno ricerca di un possibile candidato. Tra i candidati trovati, se ne esiste almeno uno, viene scelto il migliore.
    - Creazione della catena di coreferenza, basandosi anche sulle informazioni ottenute dall'analisi da parte del modulo *Orthographic Coreference*.

I primi due moduli fungono solo da moduli di supporto al terzo, generando annotazioni temporanee, in quanto non svolgono alcuna reale azione di risoluzione di coreferenza. Il modulo che svolge il reale compito di risoluzione delle coreferenze è *pronominal resolution module*.

4.4.9 Esempio di NER utilizzando ANNIE

Per chiarire meglio i meccanismi descritti e capire le potenzialità di ANNIE nell'ambito dell'IE e del NER, si prenda come esempio un'applicazione composta da tre moduli, *tokeniser*, *gazetteer* e *semantic transducer*, che deve annotare l'espressione "800,000 US *dollars*" come entità di tipo "*Number*" e sub-tipo "*Money*" (Cunningham, 2014). Di seguito è riportato un esempio di regola JAPE che permette di riconoscere entità di tipo "Money":

```
Macro:MILLION_BILLION
({Token.string=="m"}|
{Token.string=="million"}|
{Token.string=="b"}|
{Token.string=="billion"}
)


Macro:AMOUNT_NUMBER
({Token.kind==number}
(({Token.string==","}|
Token.string=="."})
{Token.kind==number})*
(({SpaceToken.kind==space})?
(MILLION_BILLION)?)
)


Rule: Money1
//e.g.30 pounds
(
(AMOUNT_NUMBER)
(SpaceToken.kind==space)?
({Lookup.majorType==currency_unit})
)
:money -->
```

```
:money.Number={kind="money",rule="Money1"}
```

A questo punto possiamo suddividere il processo di annotazione in tre fasi, ognuna svolta da un preciso componente:

1. *Tokenization*: in questa fase il testo viene separato in *token*, e a ogni *token* è associato un attributo "tipo" ("*kind"*) e la dimensione in numero di caratteri ("*length*").

```
Token, string = '800', kind = number, length = 3
Token, string = ',', kind = punctuation, length = 1
Token, string = '000', kind = number, length = 3
SpaceToken, string = ' ', kind = space, length = 1
Token, string = 'US', kind = word, length = 2, orth =
allCaps
SpaceToken, string = ' ', kind = space, length = 1
Token, string = 'dollars', kind = word, length = 7, orth
= lowercase
```

2. *List Lookup*: *gazetteer* in questa fase cerca una corrispondenza tra le stringhe nel testo e le entry di varie *lookup list.* Viene trovata la seguente corrispondenza per la stringa "US *dollars*".

```
Lookup, minorType = post_amount, majorType =
                    currency_unit
```

3. *Grammar rules*: nell'ultima fase viene applicata la regola JAPE per il riconoscimento di entità di tipo "*Money*" presentata in precedenza. La macro MILLION_BILLION è in grado di riconoscere le stringhe "m", "*million*", "b", "*billion*", ma dato che nel testo non è presente nessuna di queste stringhe si passa alla macro successiva, AMOUNT_NUMBER.

AMOUNT_NUMBER è in grado di riconoscere stringhe numeriche, opzionalmente separate da stringhe "." o ",", opzionalmente seguite da uno spazio ed opzionalmente seguite da una delle espressioni che possono essere individuate dalla macro MILLION_BILLION. Nel nostro caso verrà riconosciuto il valore numerico "800,000" come corrispondente al pattern della macro AMOUNT_NUMBER. Infine verrà applicata la regola Money1; Money1 ha il compito di identificare una stringa di tipo AMOUNT_NUMBER, seguita da uno spazio opzionale, seguita da una entità identificata come "*currency_unit*". In questo caso la stringa "800.000" era già stata riconosciuta come AMOUNT_NUMBER, è presente subito di seguito uno spazio ("*SpaceToken"* di tipo "*space"*) ed è presente un'entità già identificata dal componente *gazetteer* come "*currency_unit*", dunque viene associata l'annotazione:

```
Number, kind = money, rule = Money1
```

all'intera stringa "800,000 US *dollars*".

5. Conclusioni

In questo lavoro, in primo luogo, si è analizzato il contesto in cui è nato NLP e le ragioni che ne hanno promosso e accelerato lo sviluppo, per poi proseguire con la descrizione delle attività che NLP comprende. Tra queste attività è stato approfondito il processo di IE, spiegando gli scenari applicativi, la suddivisione in sub-task e la differenza rispetto a IR. Si è entrato poi nel merito di un sub-task di IE, NER, e sono stati analizzati in modo approfondito l'architettura e i *tool* messi a disposizione dal software di LP GATE/ANNIE, parlando delle sue funzionalità come IDE, per poi descrivere le caratteristiche e le potenzialità del linguaggio di programmazione JAPE. Infine si è conclusa la panoramica su GATE descrivendone i componenti principali.

NER non è ancora un problema risolto: i buoni risultati ottenuti negli anni passati sono dipesi dalla limitatezza dei domini considerati e dall'esiguo numero di documenti che componevano i corpora (Mónica Marrero, 2013). Una sfida futura che i ricercatori dovranno affrontare sarà sicuramente quella di trovare soluzioni per aumentare la portabilità dei sistemi NER mantenendo bassi i costi. Le attuali tecniche di *supervised machine learning* sono efficienti nell'adattare un sistema ad un nuovo dominio applicativo, ma richiedono un elevato numero di documenti annotati di training, spesso non disponibili o comunque costosi da ottenere. Per questo le ricerche nel campo di *machine learning* in ambito NER si stanno concentrando su tecniche di tipo *semisupervised* o *unsupervised*, che utilizzano documenti non annotati o solo in parte annotati per il training.

Nonostante queste problematiche, NER diverrà sempre più importante nei prossimi anni: i sistemi NER sono già attualmente utilizzati con successo nel campo della genetica, della biologia e della finanza, ma il loro uso aumenterà soprattutto per l'analisi di documenti testuali generati da organizzazioni e privati ogni giorno, ad esempio sul Web o sui social network, con un forte impatto sulla società del futuro.

Bibliografia

Cunningham. (2014). *Developing Language Processing Components with GATE Version 8.* University of Sheffield Department of Computer Science.

Dale, R. (2010). Classical Approaches to Natural Language Processing. In F. J. Nitin Indurkhya, *Handbook of Natural Language Processing, 2nd ed.* (p. 4). CRC Press.

David Nadeau, S. S. (2006). A survey of named entity recognition and classification. In E. R. Satoshi Sekine, *Named Entities: Recognition, classification and use* (pp. 3-26). Lingvisticæ Investigationes.

Hobbs, J. R. (1997). Fastus. In Y. S. Emmanuel Roche, *Finite-state Language Processing* (p. 386).

Hobbs, J. R. (2010). Information Extraction. In F. J. Nitin Indurkhya, *Handbook of Natural Language Processing, 2nd ed.* (p. 511). CRC Press.

Hutchins. (1986). ALPAC: The (In)Famous Report. In Hutchins, *Readings in Machine Translation* (pp. 131-132). Ellis Horwood Ltd.

Jakub Piskorski, R. Y. (2012). Information Extraction: Past, Present and Future. In S. H. Poibeau T, *Multi-source, Multilingual Information Extraction and Summarization* (p. 24). Springer.

Leon Derczynski, D. M. (2014). *Analysis of named entity recognition and linking for tweets.* Elsevier Ltd.

Liddy, E. D. (2001). Encyclopedia of Library and Information Science, 2nd Ed. New York: Marcel Decker.

Mónica Marrero, J. U.-C.-B. (2013). *Named Entity Recognition: fallacies, challenges and opportunities.* Madrid: University Carlos III of Madrid.

Named Entity Scores - English. (2001). Retrieved from National Institute of Standards and Technology: http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/ne_english_score_report.html

Treccani. (2015). *Dizionario Treccani.* Treccani.