UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

Studenti

Claudio Dotani N46/2878
Giuseppe Ferrara N46/2766
Gianfranco Foscardi N46/2256
Fabrizio Cappella N46/2872

Elaborato di Sistemi Multimediali

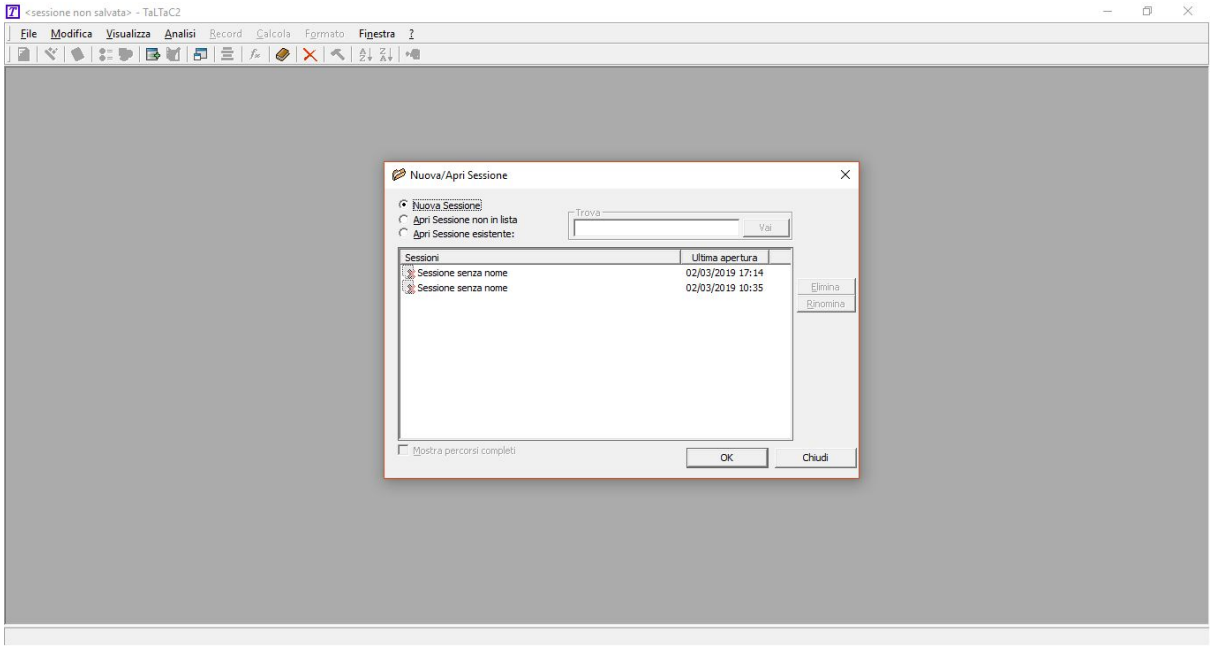
Trattamento del testo

Il trattamento del testo è una tecnica che consiste nell'estrarre automaticamente informazioni da testi. Le informazioni estratte non sono solo le singole parole, ma la semantica, che è data dal contesto in cui queste parole sono utilizzate.
Questo processo consente poi l'accesso alle informazioni contenute nei testi effettuando una ricerca basata su contenuto e non solo su parole chiave. La ricerca per contenuto richiede che le informazioni contenute nel testo debbano essere estratte e poi formalizzate.

Le principali problematiche però sono quelle di poter trovare dei risultati “inutili” perché una stessa parola può avere due accezioni in ambiti completamente differenti, ad esempio “indice” può essere inteso come il dito della mano o come quello presente nei libri di testo.
Quindi è possibile che nella ricerca compaiano dei documenti irrilevanti oppure che non vengano inclusi documenti importanti.
Grazie alle applicazioni di trattamento del testo, come quelle che abbiamo provato, *GATE* e *TALTAC*^{2} , è possibile effettuare ricerche basate non solo su parole chiave, ma anche su sinonimi della parola da ricercare oppure effettuare interrogazioni in linguaggio naturale.

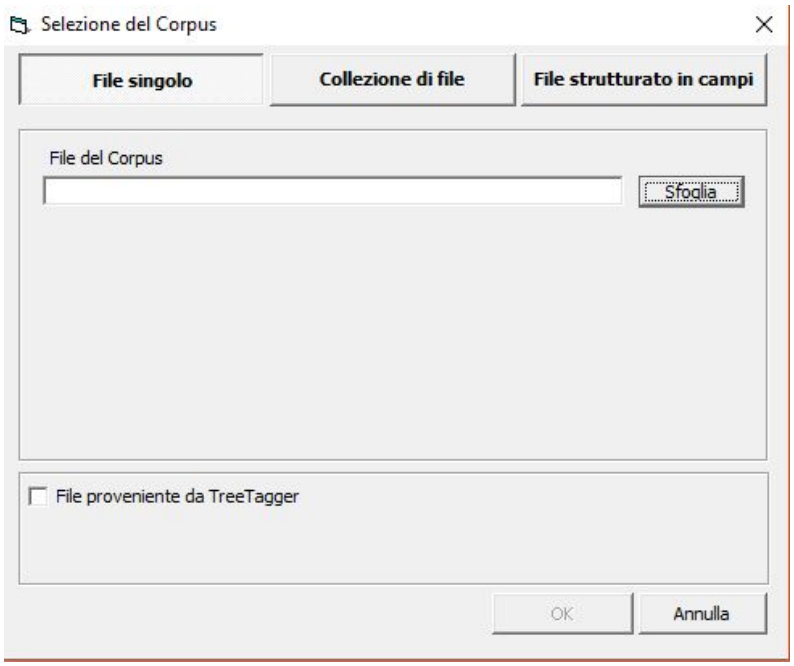
Taltac

TALTAC^2 [1] è uno strumento che consente di effettuare l’analisi e il trattamento di testi.
In generale la fase 0 di un’analisi del testo, che non sempre deve essere eseguita, è la digitalizzazione del documento da analizzare, possibile grazie all’utilizzo di scanner o fotocamere.
Sfortunatamente abbiamo avuto accesso solo alla versione DEMO del programma, quindi non abbiamo potuto testarne tutte le funzionalità.
Per poter effettuare l’analisi di un testo è necessario prima creare una sessione di lavoro.

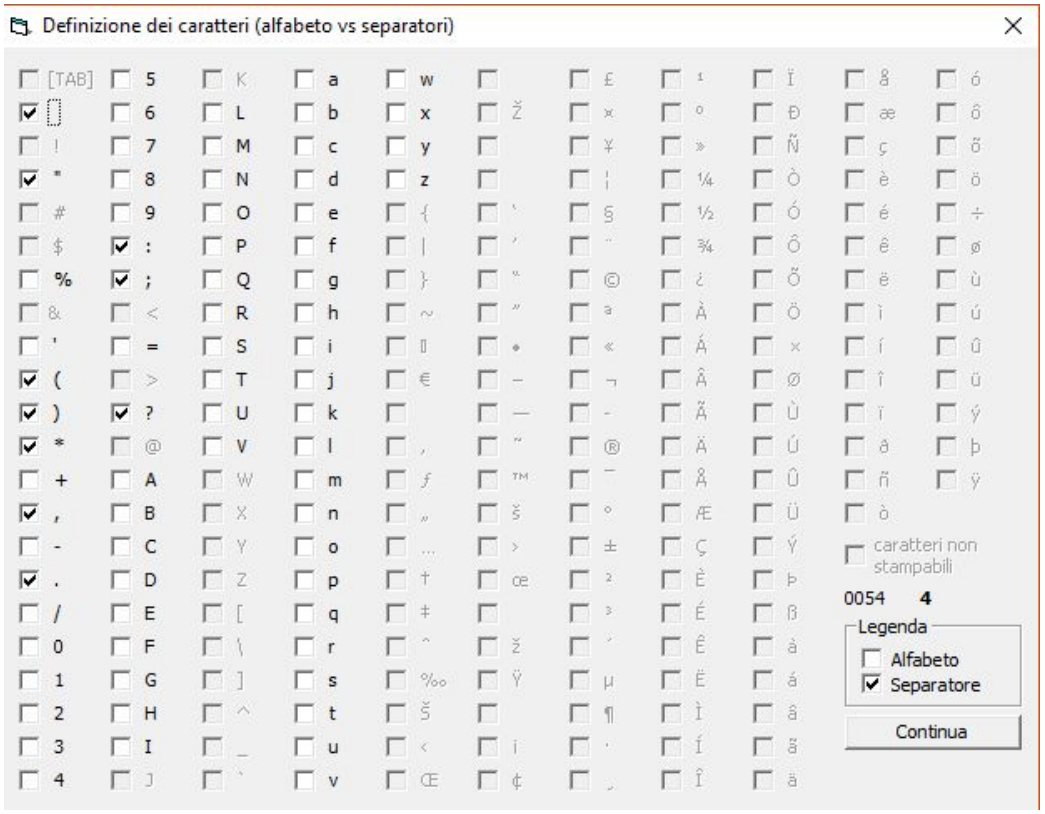


Una volta creata è poi possibile scegliere il file da analizzare. In questo caso si è presupposto che il documento sia già stato digitalizzato oppure che sia stato generato direttamente in formato digitale.

[1] **TALTAC2** : **T**rattamento **A**utomatico **L**essicale e **T**estuale per l'**A**nalisi del **C**ontenuto di un **C**orpus.

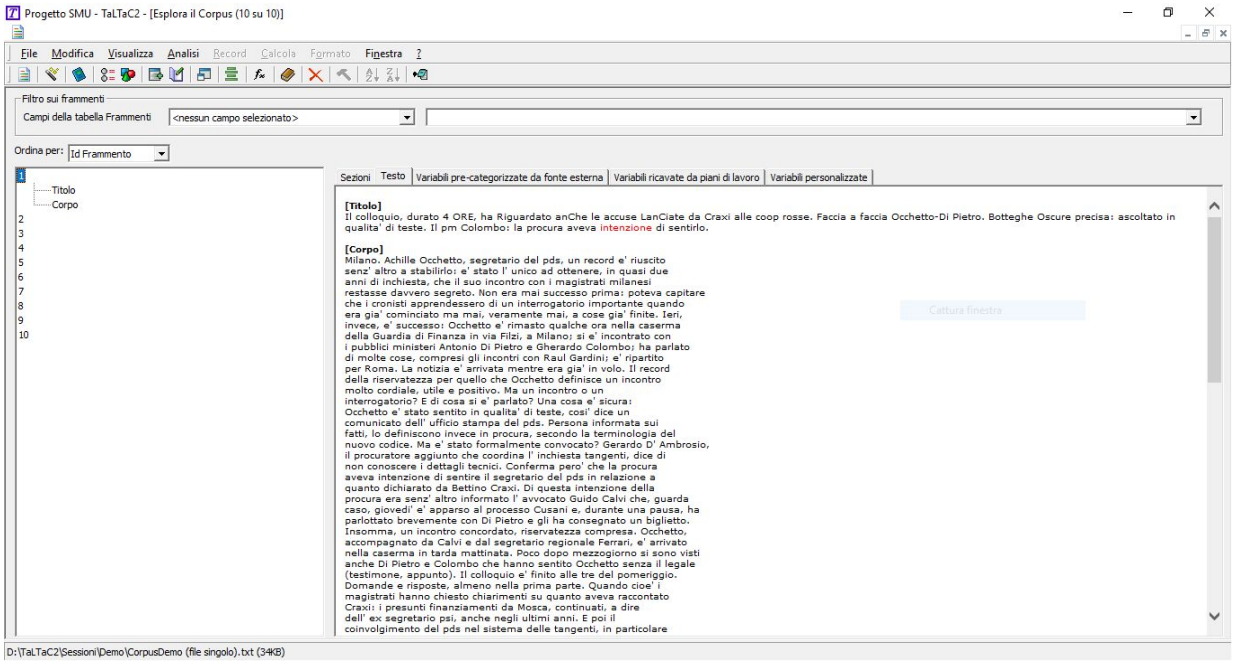


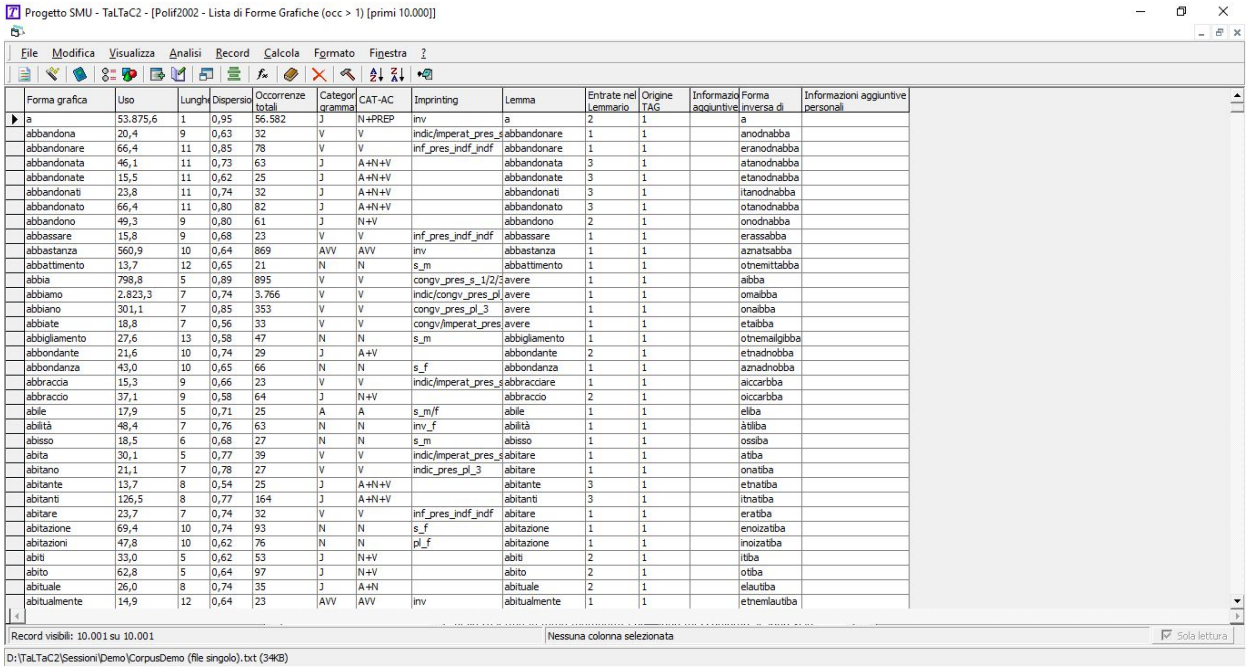
Dopo aver selezionato il documento in versione digitale viene richiesto se eseguire il parsing del documento, ovvero una tecnica di pretrattamento del testo, che consente di definire i due insiemi di caratteri di un testo, i separatori e i caratteri dell'alfabeto.



Durante questa fase si definiscono i due insiemi fondamentali di caratteri di un testo, i separatori e i caratteri dell'alfabeto. Questi insiemi sono duali, quindi se un carattere è utilizzato come separatore non verrà mai considerato come carattere di un alfabeto.

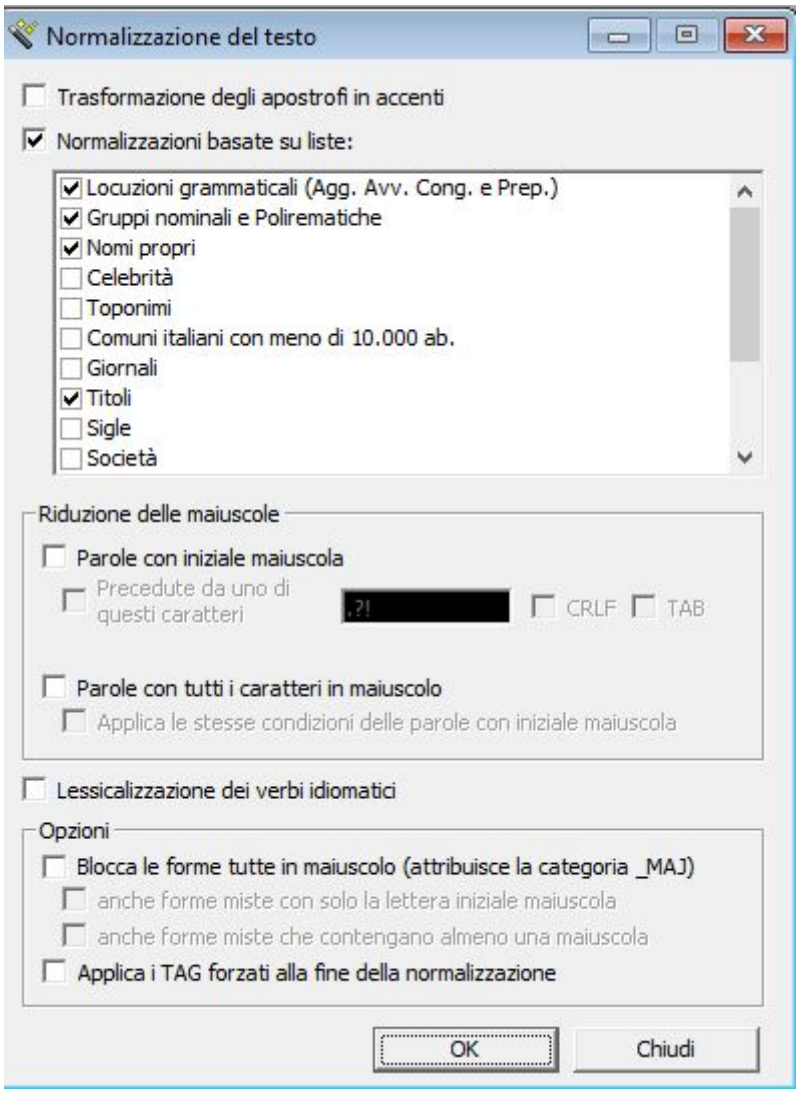
Successivamente avviene il riconoscimento dei token, cioè la sequenza di caratteri dell'alfabeto compresi tra due caratteri separatori.
TALTAC prevede tre ambienti, il DB di Sessione, che contiene le liste e/o tabelle prodotte in fase di analisi, un ambiente che consente di esplorare il corpus del documento e un insieme di risorse **statistico-linguistiche** che possono essere confrontate con il testo.





Forma grafica	Uso	Lunghe	Dispersio	Occorrenze totali	Categor gramma	CAT-AC	Imprinting	Lemma	Entrate nel Lemmario	Origine TAG	Informazio aggiuntive	Forma inversa di	Informazioni aggiuntive personali
a	53.875,6	1	0,95	56.582	J	N+PREP	inv	a	2	1		a	
abbandona	20,4	9	0,63	32	V	V	indic/imperat_pres_s	abbandonare	1	1		anodnabba	
abbandonare	66,4	11	0,85	78	V	V	inf_pres_indf_indf	abbandonare	1	1		eranodnabba	
abbandonata	46,1	11	0,73	63	J	A+N+V		abbandonata	3	1		atanodnabba	
abbandonate	15,5	11	0,62	25	J	A+N+V		abbandonate	3	1		etanodnabba	
abbandonati	23,8	11	0,74	32	J	A+N+V		abbandonati	3	1		itanodnabba	
abbandonato	66,4	11	0,80	82	J	A+N+V		abbandonato	3	1		otanodnabba	
abbandono	49,3	9	0,80	61	J	N+V		abbandono	2	1		onodnabba	
abbassare	15,8	9	0,68	23	V	V	inf_pres_indf_indf	abbassare	1	1		erassabba	
abbastanza	560,9	10	0,64	869	AVV	AVV	inv	abbastanza	1	1		aznatsabba	
abbattimento	13,7	12	0,65	21	N	N	s_m	abbattimento	1	1		otnemittabba	
abbia	798,8	5	0,89	895	V	V	congv_pres_s_1/2/3	avere	1	1		aibba	
abbiamo	2.823,3	7	0,74	3.766	V	V	indic/congv_pres_pl	avere	1	1		omaibba	
abbiano	301,1	7	0,85	353	V	V	congv_pres_pl_3	avere	1	1		onaibba	
abbiate	18,8	7	0,56	33	V	V	congv/imperat_pres	avere	1	1		etaibba	
abbigliamento	27,6	13	0,58	47	N	N	s_m	abbigliamento	1	1		otnemailgibba	
abbondante	21,6	10	0,74	29	J	A+V		abbondante	2	1		etnadnobba	
abbondanza	43,0	10	0,65	66	N	N	s_f	abbondanza	1	1		aznadnobba	
abbraccia	15,3	9	0,66	23	V	V	indic/imperat_pres_s	abbracciare	1	1		aiccarbba	
abbraccio	37,1	9	0,58	64	J	N+V		abbraccio	2	1		oiccarbba	
abile	17,9	5	0,71	25	A	A	s_m/f	abile	1	1		elba	
abilità	48,4	7	0,76	63	N	N	inv_f	abilità	1	1		ătiliba	
abisso	18,5	6	0,68	27	N	N	s_m	abisso	1	1		ossiba	
abita	30,1	5	0,77	39	V	V	indic/imperat_pres_s	abitare	1	1		atiba	
abitano	21,1	7	0,78	27	V	V	indic_pres_pl_3	abitare	1	1		onatiba	
abitante	13,7	8	0,54	25	J	A+N+V		abitante	3	1		etnatiba	
abitanti	126,5	8	0,77	164	J	A+N+V		abitanti	3	1		itnatiba	
abitare	23,7	7	0,74	32	V	V	inf_pres_indf_indf	abitare	1	1		eratiba	
abitazione	69,4	10	0,74	93	N	N	s_f	abitazione	1	1		enoizatiba	
abitazioni	47,8	10	0,62	76	N	N	pl_f	abitazione	1	1		inoizatiba	
abiti	33,0	5	0,62	53	J	N+V		abiti	2	1		itiba	
abito	62,8	5	0,64	97	J	N+V		abito	2	1		otiba	
abituale	26,0	8	0,74	35	J	A+N		abituale	2	1		elautiba	
abitualmente	14,9	12	0,64	23	AVV	AVV	inv	abitualmente	1	1		etnemlautiba	

Successivamente al parsing avviene la tecnica di normalizzazione, che, utilizzando apposite basi dati e regole, elimina i “doppioni” e le ambiguità. Potrebbero essere presenti infatti parole identiche ma scritte in modo leggermente differente, ad esempio una sigla puntata o con l’acronimo scritto interamente in maiuscolo possono essere considerate la stessa parola (T.A.L.T.A.C. = TALTAC).



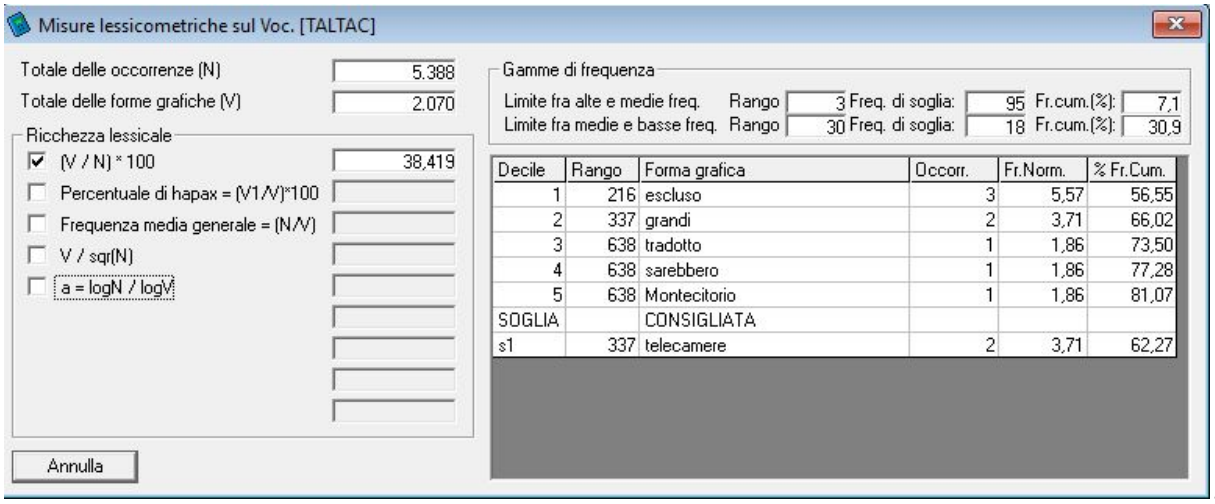
Gli strumenti di analisi statistico-lessicale messi a disposizione dal software sono i seguenti:

- calcolo delle misure lessicometriche
- analisi dei segmenti
- tagging grammaticale
- analisi delle concordanze
- imprinting
- lemmatizzazione

Calcolo delle misure lessicometriche

Taltac consente di eseguire il calcolo di indici statistici sul dizionario e le classi di frequenza delle parole nel testo.
Dato il corpus di un testo è possibile definire diversi valori come la grandezza del vocabolario, la lunghezza in parole e il loro rapporto, la ricchezza lessicale.
La prima è data dal numero di parole differenti contenute nel testo, la seconda invece è data dal numero delle parole (o token) contenute nel corpus, la terza invece indica il rapporto di

queste due grandezze. Il rapporto viene utilizzato per avere un'idea della grandezza del vocabolario, infatti un valore vicino allo zero indica che i termini utilizzati nel testo sono quasi sempre i medesimi, mentre un valore prossimo ad uno indica un lessico molto vario.
Se una parola intercorre una volta sola in un testo si chiama **hapax**, la cui percentuale è definita mediante il rapporto del numero di questi con la grandezza del vocabolario



Analisi dei segmenti

Questa tecnica ha lo scopo di individuare i segmenti, ovvero blocchi di parole semanticamente coesi ed allo stesso tempo di selezionare quelli più rilevanti.
I segmenti vengono individuati mediante varie tecniche, ad esempio

- utilizzando termini tecnici
- costruzioni a verbo supporto
- nomi propri composti
- espressioni idiomatiche

per un segmento è definito il numero massimo di parole che lo compongono, i separatori forti, la soglia di frequenza minima delle parole e del segmento stesso nel corpus.
Per rilevare automaticamente i segmenti si utilizza l'indice di significatività, che è un valore che indica quanto spesso due parole sono presenti insieme, tanto più è alto questo valore tanto più è la probabilità che queste parole formino un segmento.
Taltac utilizza però l'indice di significatività relativo, perchè quello assoluto è condizionato dal numero di parole che compongono il segmento.
Una volta individuati i segmenti si procede alla lessicalizzazione, cioè si trasformano i segmenti in parole uniche.

Tagging grammaticale

Questa è una procedura che consiste nel classificare ogni parola o token come Avverbio, Aggettivo, Pronome ecc... in modo da aumentare la precisione nel recupero delle informazioni.

Estrazione Di Informazioni

L'estrazione di informazione del corpus oggetto di analisi si propone di identificare unità lessicali semplici e complesse espressioni di entità di interesse del dominio.

Tali unità possono essere:

- *Forme lessicali rilevanti*: si tratta delle parole chiave del corpus, ovvero dei termini semanticamente discriminanti in quanto descrittori dei contenuti del corpus o dei singoli frammenti testuali. Le keywords possono coincidere con i termini di dominio, ma non è sempre così;
- *Forme lessicali peculiari*: si tratta delle unità lessicali tipiche del corpus oggetto di analisi, tipiche in quanto originali o molto specifiche.

Calcolo dell'indice TFIDF

- Una prima strategia per l'estrazione del linguaggio peculiare è il calcolo dell'indice TF IDF (*Term Frequency – Inverse Document Frequency*). L'indice TFIDF rappresenta un peso attribuito a ciascuna parola sulla base della sua frequenza e della sua distribuzione all'interno della collezione dei documenti, ed è questo peso che viene preso in considerazione per effettuare l'ordinamento dei risultati.
- L'indice TFIDF è espresso dalla seguente ponderazione:

$$Wtd = f\,td * \log N/ft$$

Dove Wtd è il peso del termine t nel documento d, ftd è la frequenza del termine t nel documento d, N è il numero totale di documenti nel corpus e ft è il numero di documenti contenenti questo termine.

- L'indice si basa su due assunti fondamentali:

tanto più un termine occorre in un documento tanto più è rappresentativo del suo contenuto;

tanti più documenti contengono un termine, tanto meno questo è discriminante.

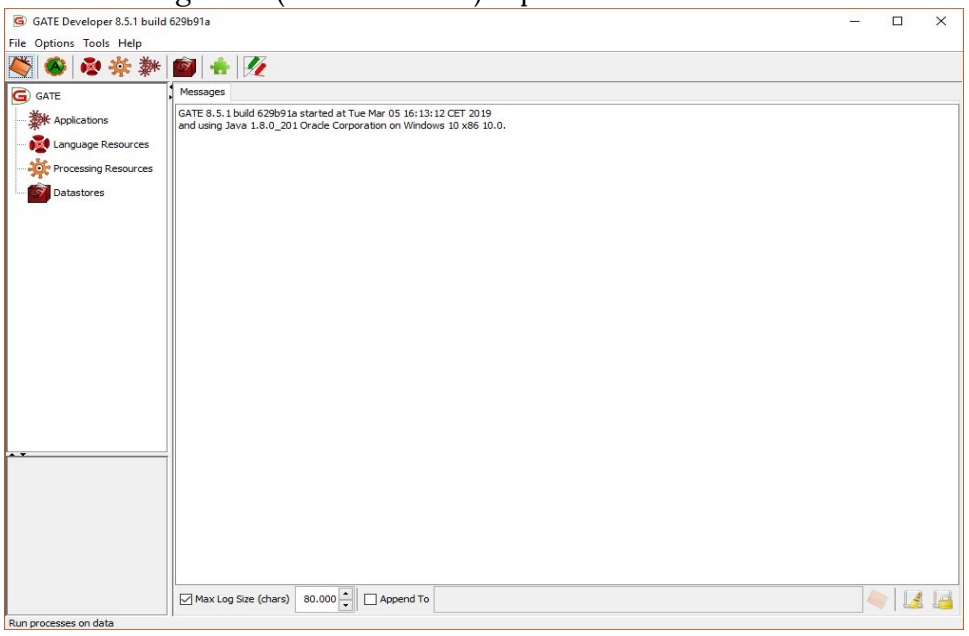
Gate

Poichè il software precedentemente descritto è disponibile gratuitamente solo in versione demo, con funzionalità limitate, abbiamo utilizzato Gate, un altro software di analisi e trattamento del testo, che nella sua versione **GATE Developer** è con licenza gratuita (**LGPL**).

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 Language Processing (LP). Esso è contemporaneamente un'architettura, nel senso che definisce un'organizzazione ad alto livello dei sistemi per LP e assicura una corretta interazione tra i componenti. Può essere considerato anche un **Integrated Development Environment (IDE)** perchè aiuta gli utenti a sviluppare in modo semplice ed efficiente applicazioni per NLP, fornendo una GUI e funzionalità per il debugging un Framework. Esiste anche una Web App poiché permette di annotare collettivamente un set di documenti

La versione di Gate da noi utilizzata è la **Developer,** il software fornisce un ambiente grafico per lo sviluppo di software dedicato al trattamento del testo. Lo scopo principale diè l'annotazione dei documenti, quindi tutte le sue funzionalità sono relative a questo scopo. Vi è incluso un sistema completo di information extraction chiamato ANNIE[2], spesso utilizzato per creare una rappresentazione RDF o OWL per i contenuti non strutturati.

L'interfaccia grafica (Main Window) si presenta così:



- In alto a sinistra: albero delle risorse.
- In basso a sinistra: anteprima delle risorse.
- Al centro: viewer principale delle risorse.
- In Basso: la barra dei messaggi.

Documenti supportati

- I formati di documenti supportati da GATE automaticamente sono: file di testo, **HTML, SGML, XML, RTF**, alcuni file **PDF**, **email**, alcuni **formati di Microsoft Office e di OpenOffice**.

[2] ANNIE: A Nearly-New Information Extraction System

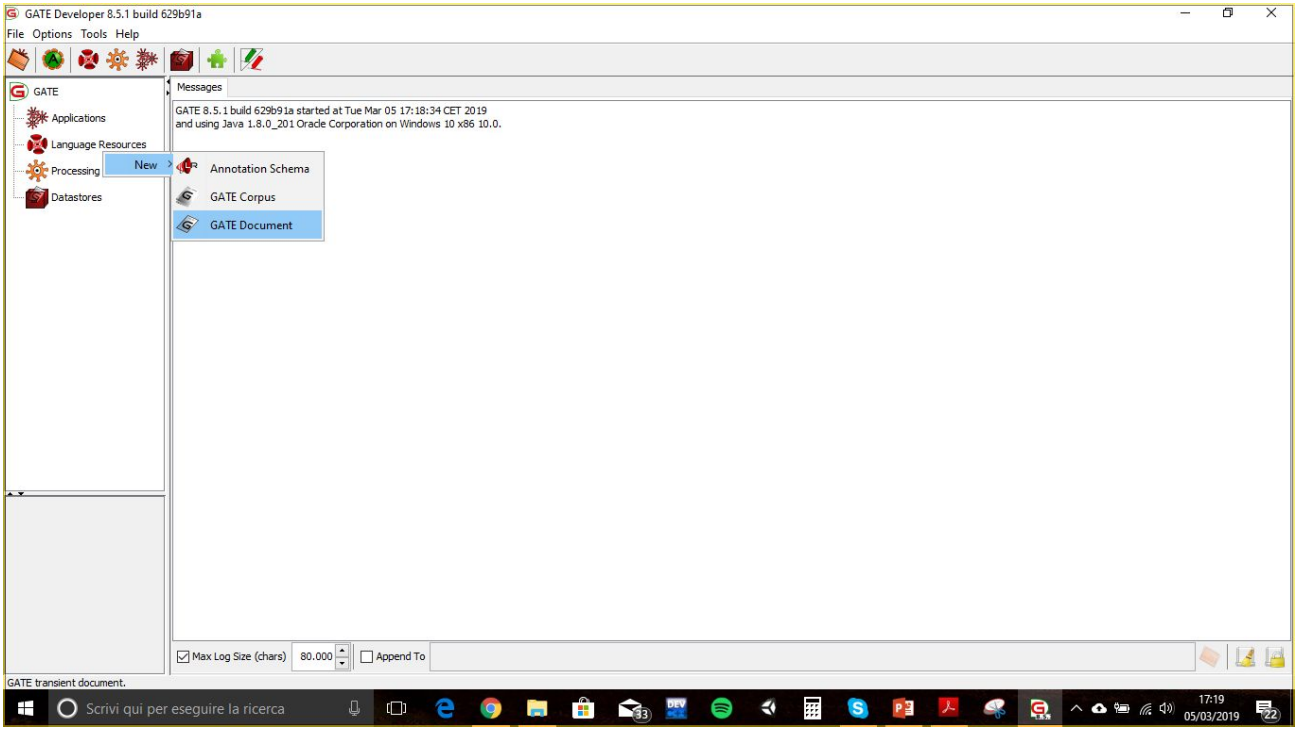
- Possibilità di utilizzo di altri formati particolari tramite l'uso di plugin (ad esempio per file **JSON** contenti tweet o per **markup MediaWiki**, utilizzati da Wikipedia e altri siti).

- GATE è in grado di gestire file testuali strutturati o semistrutturati associando al documento le informazioni in esso contenute sotto forma di annotazioni.

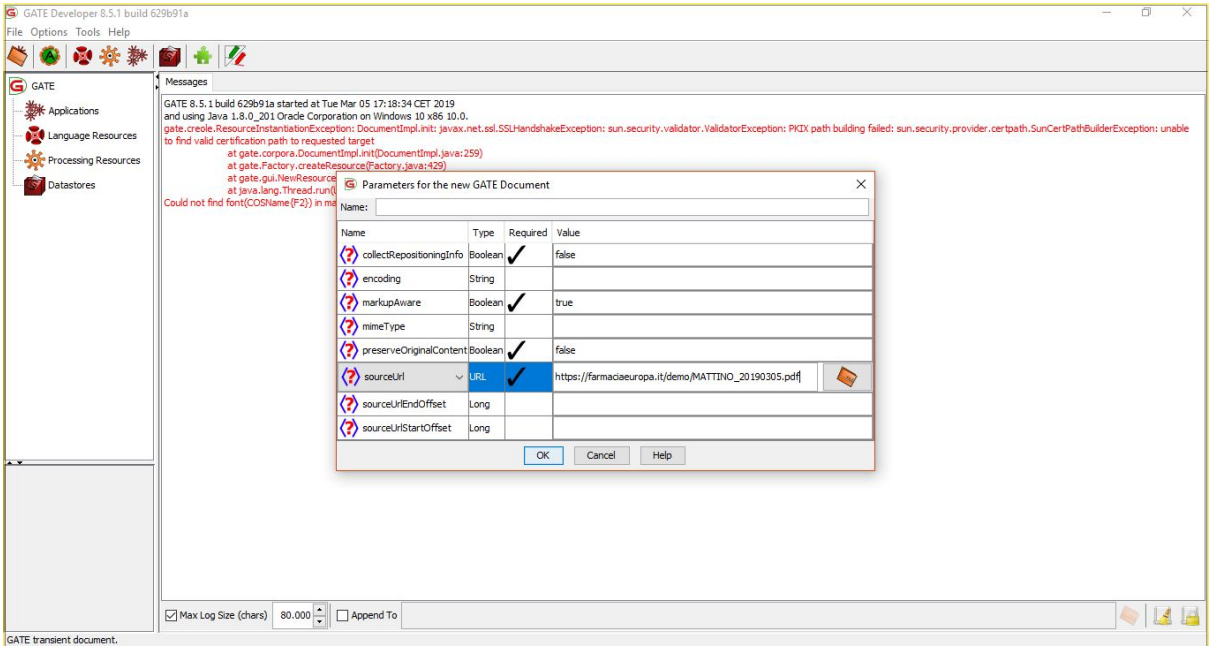
- Al momento dell'importazione di un documento, GATE ricerca all'interno del contenuto l'eventuale presenza di tag, che, qualora individuati, saranno rappresentati da annotazioni create ad hoc il cui tipo corrisponderà al nome del tag e le cui feature saranno estratte dagli attributi dello stesso.

- I documenti possono essere esportati o salvati in un Data Store, per usi futuri in GATE Developer e altri componenti della suite GATE

Creazione del Corpus e inserimento dati



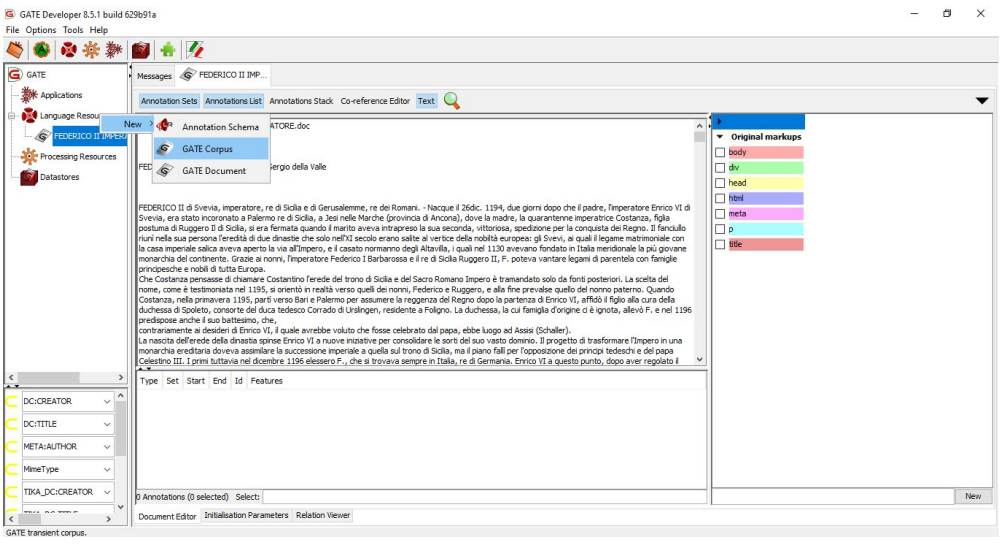
Per mezzo dell'immagine sovrastante si può notare il metodo di caricamento dei documenti attraverso degli url. nel nostro caso abbiamo utilizzato questo Link.

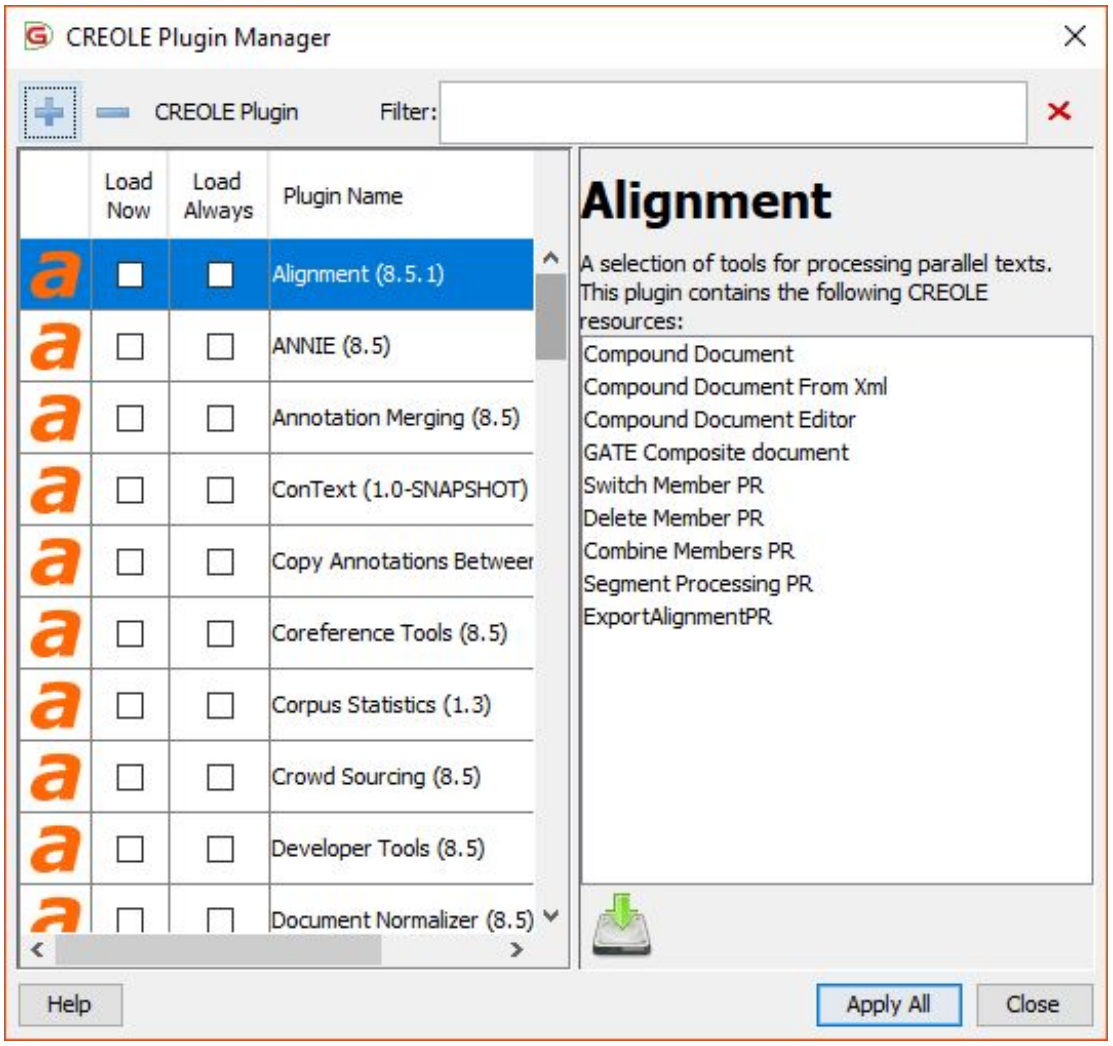


Quest' operazione ci ha consentito di caricare un file .pdf da un URL. Una volta aperto il file abbiamo cliccato su **Annotation Sets e Annotations List**, cosa che ci ha permesso di identificare, quindi filtrare le annotazioni.

- **Annotation Sets ->** Vengono filtrate le annotazioni divise per tipo, cliccando sulla rispettiva casella.
- **Annotation List ->** Viene filtrato l'elenco delle annotazioni specifiche, che sono evidenziate a mezzo di finestre colorate.

In seguito abbiamo esplorato il **Gate Corpus** creando una nuova **Language Resource** esplorando così i **CREOLE plugins.**





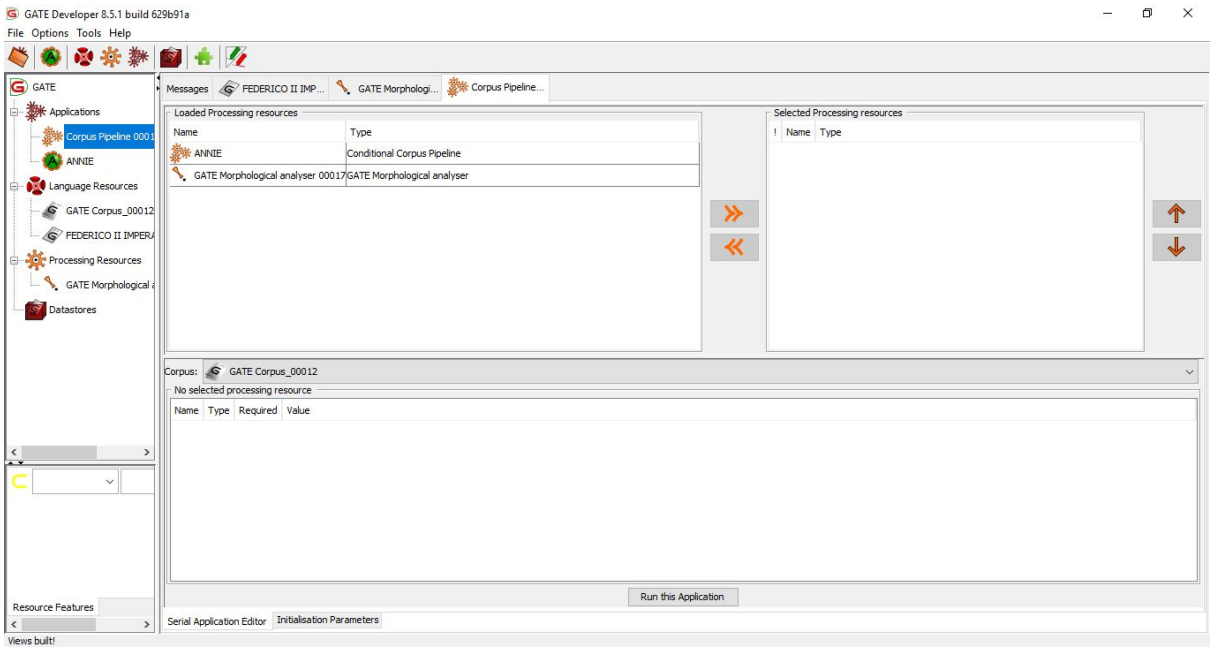
Provando il Plugin ANNIE abbiamo scoperto che contiene le seguenti risorse:

Tokeniser, Sentence Splitter, POS tagger, Gazetteers, Named Entity tagger (JAPE transducer), Orthomatcher (ortho-graphic coreference) e altri.

Ora applichiamo una risorsa **"Gate Morphological Analyser"** al documento**.**

- Ciascuna Processing Resource ha due insieme di parametri.
- Parametri impostati a run-time.
- Parametri impostati in fase di inizializzazione, possono essere modificati solo in fase di creazione dell'istanza della risorsa.
- Talvolta, in alcune risorse particolari (es ANNIE Gazzetteer), questi ultimi possono essere modificati dopo la creazione.

Dopo aver caricato tutte le risorse necessarie, è possibile creare un'applicazione



•Facendo doppio click è possibile.

•Costruire la pipeline selezionando le risorse caricate spostandole nel riquadro destro.

•Controllare l'ordine delle risorse (IMPORTANTE).

•Controllare i parametri di ciascuna risorsa.

•Eseguire l'applicazione tramite il tasto RUN.

Per ciascuna risorsa è necessario impostare i parametri di inizializzazione, Es. ANNIE English Tokenizer;

Esportare i documenti annotati da GATE

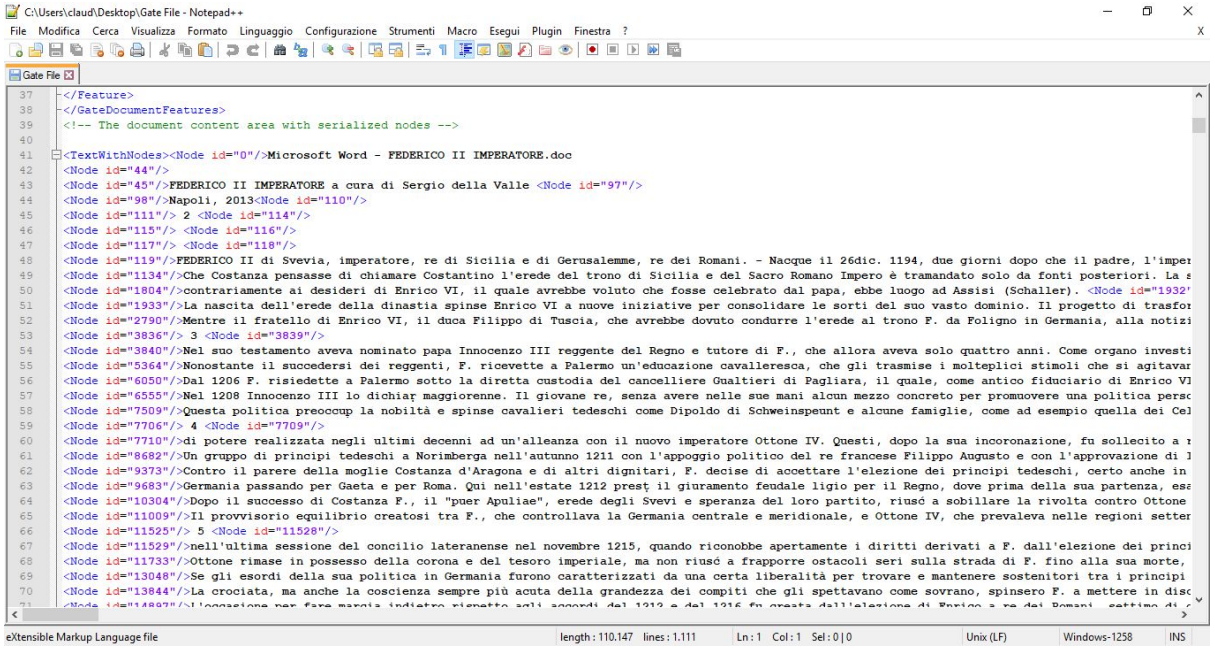
- È possibile esportare in due formati:

  GATE XML format

- Tasto destro sul documento -> Save as XML

Le annotazioni vengono aggiunte in coda al documento e ciascun tag viene annotato tramite un ID

- Preservando la formattazione originale
- Utile per file HTML o XML...



JAPE

Jape, il cui nome sta per "**J**ava **A**nnotation **P**atterns **E**ngine", è un linguaggio sviluppato per essere un componente della piattaforma GATE. JAPE opera su annotazioni basate su espressioni regolari. Pertanto è utile per l'estrazione semantica e molte altre operazioni su alberi sintattici prodotti dal parsing.

Ognuno di questi componenti (del plugin ANNIE) assume un importante ruolo nella fase di processing del testo.

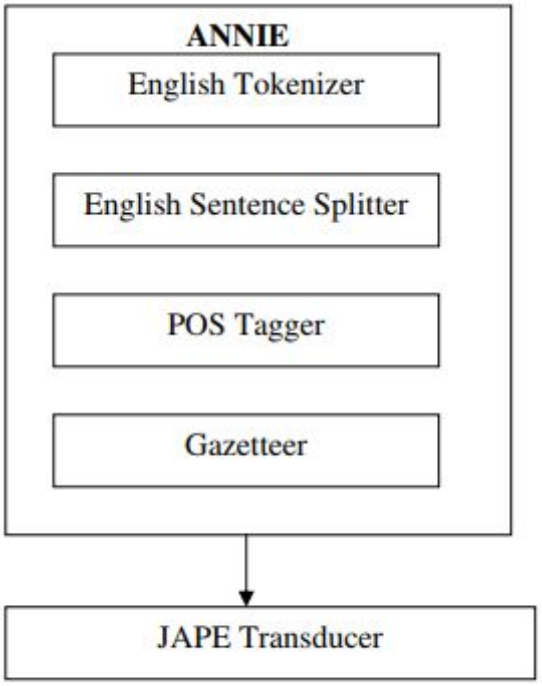


Figure 1 Typical GATE system components

il Tokenizer: divide il testo in token molto semplici (ad esempio numeri, punteggiatura, e parole di diverso tipo)

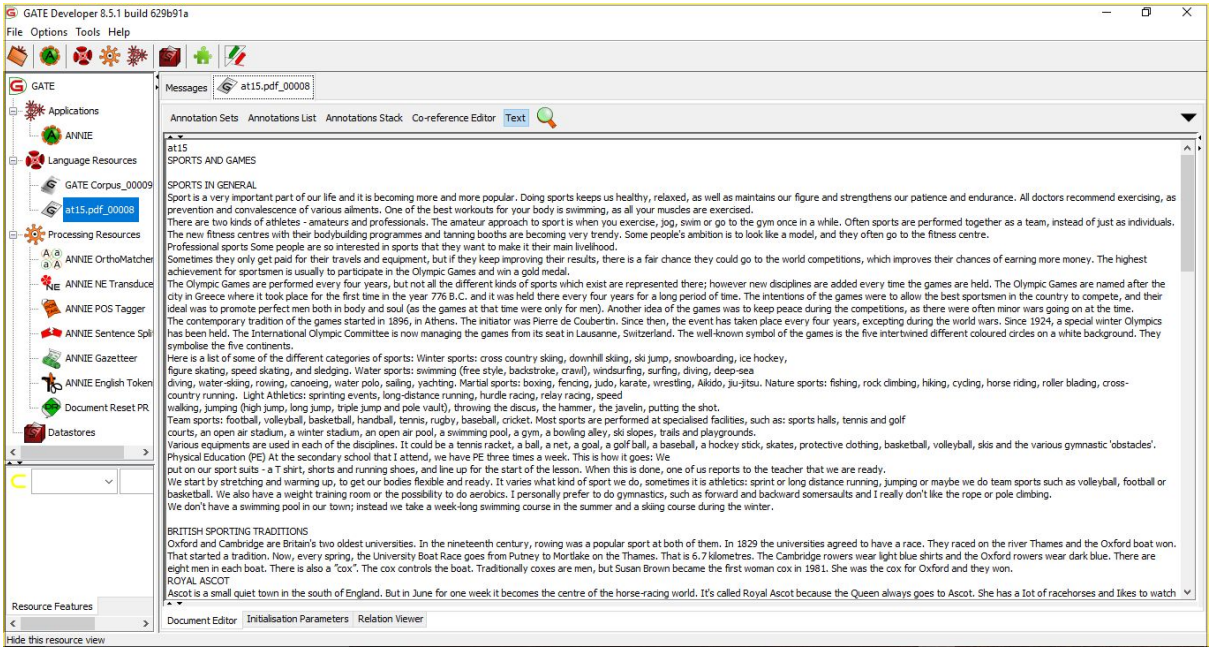
Il sentence splitter è una vera e propria cascata di stati finiti che segmentano il testo in frasi.L'output di questo modulo "funge" da input per i moduli quali : POS TARGETTER

Il Gazetteer è una lista di entità (una sorta di DB). Tali entità sono collezionate nei vari file che il Gate Gazetteer utilizzerà per "Ricercare" la fase iniziale delle annotazioni. La regola jape in particolare utilizza queste annotazioni e non solo per identificare pattern/entità presenti all'interno del testo.

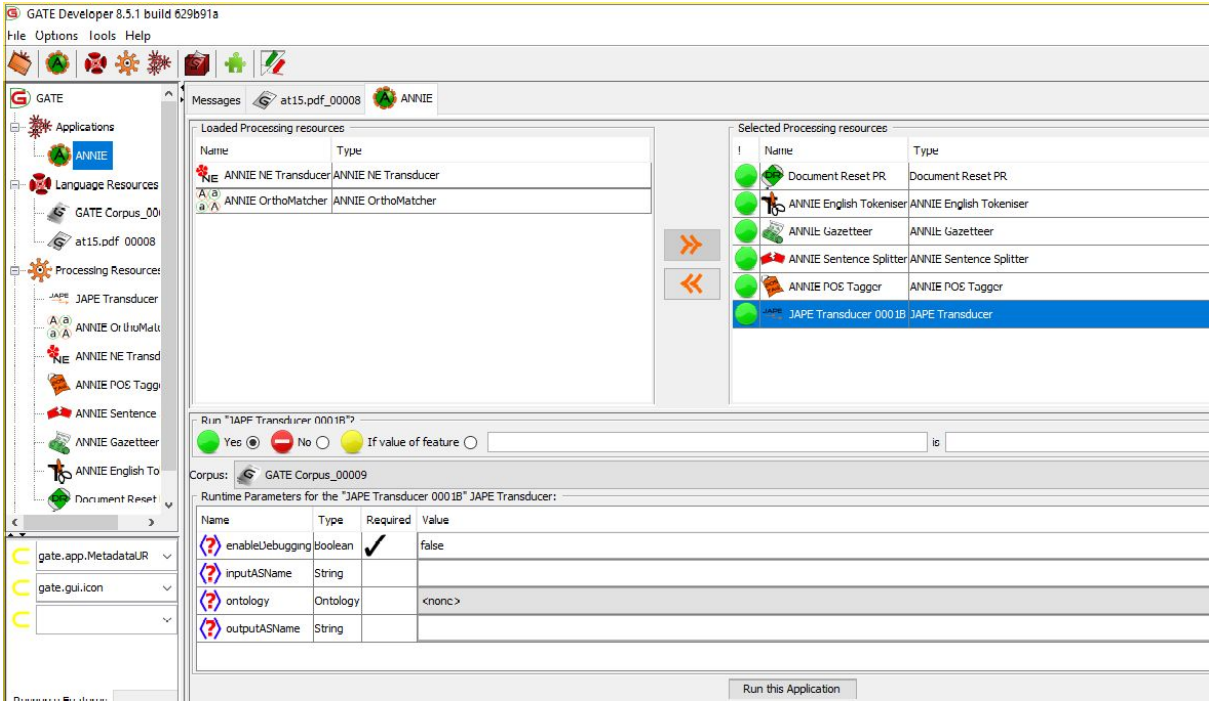
La "Grammatica" Jape consiste in un set di fasi, nelle quali vengono sfruttati pattern/ regole di azione.
La grammatica di una regola jape è sempre costituita da due fasi: la LHS o RHS (Left & Right Hand-Side)

La prima (LHS) contiene essenzialmente gli operatori (esempio *,?,+).
La seconda (RHS) determina l'azione che deve essere effettuata quando viene detectato un pattern all'interno del testo, la stessa consiste in annotazioni riguardanti gli statement manipolati.
L'annotazione consiste nel taggare, commentare o "marcare" i media con qualche metadato per identificare il tipo di contento del media stesso.Nel contesto di GATE l'annotazione consiste nel creare un nuovo oppure identificare un metadato già esistente. Per esempio identificare i nomi di persona all'interno di un testo.

Detto ciò passiamo ad un semplice esempio:
Carichiamo il file di testo preso da internet a scopo illustrativo Testo di Prova.

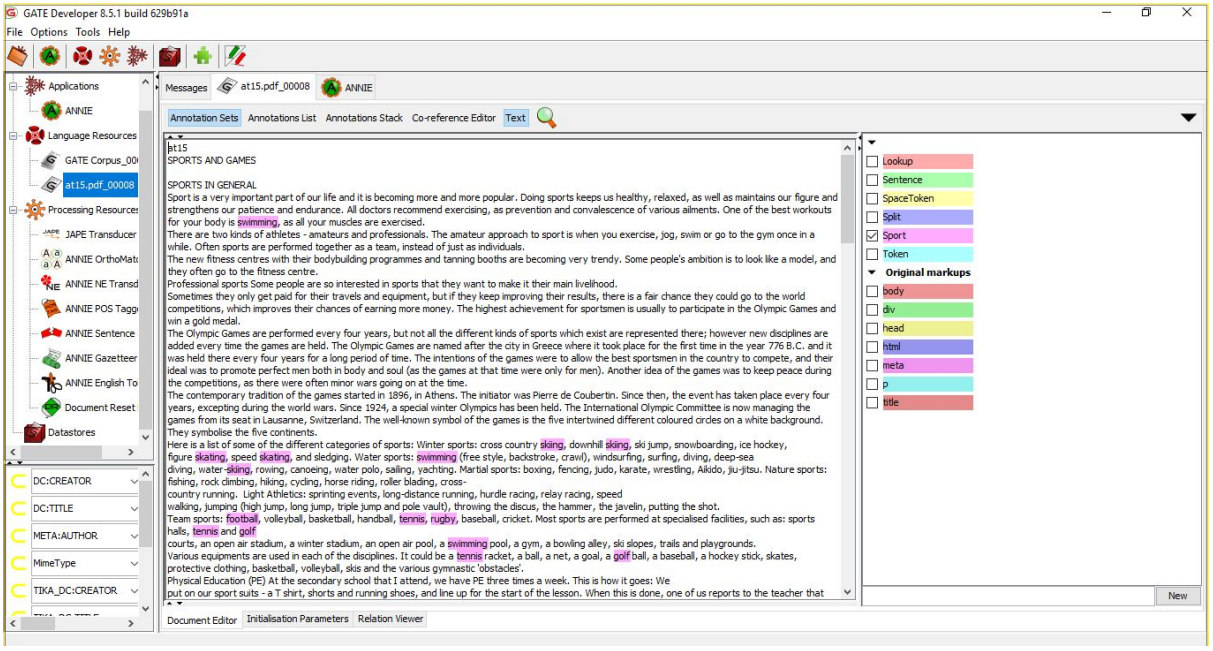


A questo punto, dopo aver caricato il testo, carichiamo il nostro file con estensione: prova.jape, nel quale risiede il codice.
I plugin **OrtoMatcher** e **Ne Trasducer** sono stati volutamente omessi dall' applicazione.
Sulla destra possiamo notare il nostro transducer.



Dall'interfaccia di GATE Noteremo ben presto che al caricamento del testo sulla destra dello schermo ci sono già delle annotazione "preimpostate". Tali annotazione sono state generate dalla lista lookup del Gazetteer.

A questo punto notiamo che il codice .jape va a creare un nuovo label di nome "Sport". Selezionando questo label, tutte le parole relative a questo Gazetteer verranno selezionate.




```
Phase: firstpass
Input: Token Lookup
Options: control = brill
Rule: SportsCategory
Priority: 20
(
{Lookup.majorType == sport}
): sportCat
→
:sportCat.Sports = {kind = "sport",rule = "SportsCategory"}
```

Linea 1: una grammatica JAPE consiste in un insieme di fasi, ognuna delle quali consiste in un insieme
di pattern / regole di azione. Queste fasi devono essere etichettate in modo univoco; qui etichettiamo la nostra fase di annotazione come firstpass. Si noti che il nome della fase può essere diverso dal nome del file di grammatica jape.

Linea 2: le annotazioni di input devono anche essere definite all'inizio di ogni grammatica, queste sono le annotazioni a cui verrà abbinata la regola. Se non sono state definite annotazioni,il valore predefinito sarà Token, Space Token e Lookup quindi, per impostazione predefinita, solo queste annotazioni saranno prese in considerazione quando si tenta di trovare qualche corrispondenza. Ogni tipo di annotazione che va ad
essere abbinata in quella fase grammaticale deve essere incluso nel set di input. Qualunque sia il tipo di annotazione che non è definito nell'input set verrà ignorato nella fase di ricerca di corrispondenze..

Linea 3: all'inizio di ogni grammatica, è possibile impostare diverse opzioni. Potrebbero essere: Controllo: definisce il metodo di corrispondenza delle regole. Le opzioni possibili sono {Appelt,Brill, All, Once}.
Debug - se impostato su true, se la grammatica è in esecuzione in modalità Appelt (opzione di controllo) e c'è più di una possibile corrispondenza, i conflitti verranno visualizzati nell'apposita finestra dei messaggi.

Linea 4: nomina una regola, in questo esempio il nome della regola è SportsCategory.
Linea 5: spiegata più avanti nell'esempio 5. Per il momento, si assuma che la priorità campo (Priorità: X) vengono utilizzati per impostare la priorità della regola rispetto ad altre regole nella stessa
file di grammatica.

Linea 4: nomina una regola, in questo esempio il nome della regola è SportsCategory.

Linea 5: Per il momento, si assuma che la priorità campo (Priorità: X) viene utilizzata per impostare la priorità della regola rispetto ad altre regole nello stesso file di grammatica.

Righe 6,7 e 8:

```
(
{Lookup.majorType == "Sport"}
): sportCat
```

Questa è una parte importante della regola. Quello che stiamo dicendo al trasduttore con questa regola è questo:
trova le annotazioni con il modello `Lookup.majortype =="Sport"` e chiamalo temporaneamente "etichetta".
La sintassi == e {} sono importanti. Il pattern che si sta cercando deve essere circondato da {} e sotto la stessa chiusura ().

Linea 9 -> è il limite della regola LHS. Ciò che segue sarà la parte RHS della regola!
Linea 10 ->

```
: sportCat.Sports = {rule = "SportsCategory"}
```

Qui stiamo dicendo al trasduttore che l'etichetta temporanea (dalla riga 8) sarà ribattezzato "Sport" e la regola che raggiunge questo è "SportsCategory". Dare un nome ad una regola qui è importante per lo scopo del debug infatti quando la stessa sarà eseguita potrà essere visualizzata nella GUI di GATE.
In questo esempio vedremo Soccer annotato come `rule = "SportsCategory"`
La sintassi della riga 10 è importante da notare, etichetta temporanea.

```
Nuova etichetta = {rule = "nome della regola"}
```

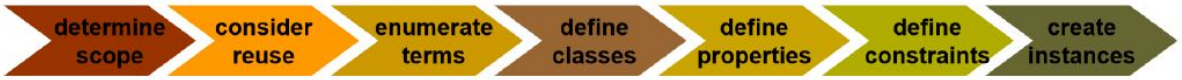
Protégé

Ontologia in Informatica:
In informatica, un'ontologia è una rappresentazione formale, condivisa ed esplicita di una concettualizzazione di un dominio di interesse. Più nel dettaglio, si tratta di una teoria assiomatica del primo ordine esprimibile in una logica descrittiva. Il termine ontologia formale è entrato in uso nel campo dell'intelligenza artificiale e della rappresentazione della conoscenza, per descrivere il modo in cui diversi schemi vengono combinati in una struttura dati contenente tutte le entità rilevanti e le loro relazioni in un dominio. I programmi informatici possono poi usare l'ontologia per una varietà di scopi, tra cui il ragionamento induttivo, la classificazione, e svariate tecniche per la risoluzione di problemi. Una ontologia fondazionale è in qualche misura assimilabile ad un glossario di base, anche se al contrario di questo, usualmente la prima è gerarchizzata in due o più livelli, nei cui termini tutto il resto deve essere descritto.

Linguaggi Per Ontologie:
Per essere utili, le ontologie devono essere espresse in una notazione concreta. Un 'linguaggio per ontologie' è un linguaggio formale con cui viene costruita un'ontologia. Esistono diversi linguaggi, proprietari o basati su standard, per la definizione di ontologie:
• OWL (Web Ontology Language)
• CycL
• IDEF5
• MOF
• OBO
• OntoUML
Il più utilizzato è OWL:

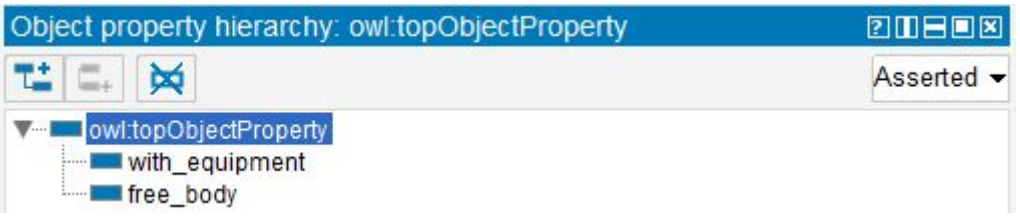
Il **Web Ontology Language (OWL)** è un linguaggio di markup per rappresentare esplicitamente significato e semantica di termini con vocabolari e relazioni tra gli stessi. Esistono varie versioni del linguaggio, che differiscono molto tra di loro.

Sviluppo di un' Ontologia
La line-up di sviluppo di un'Ontologia è la seguente:



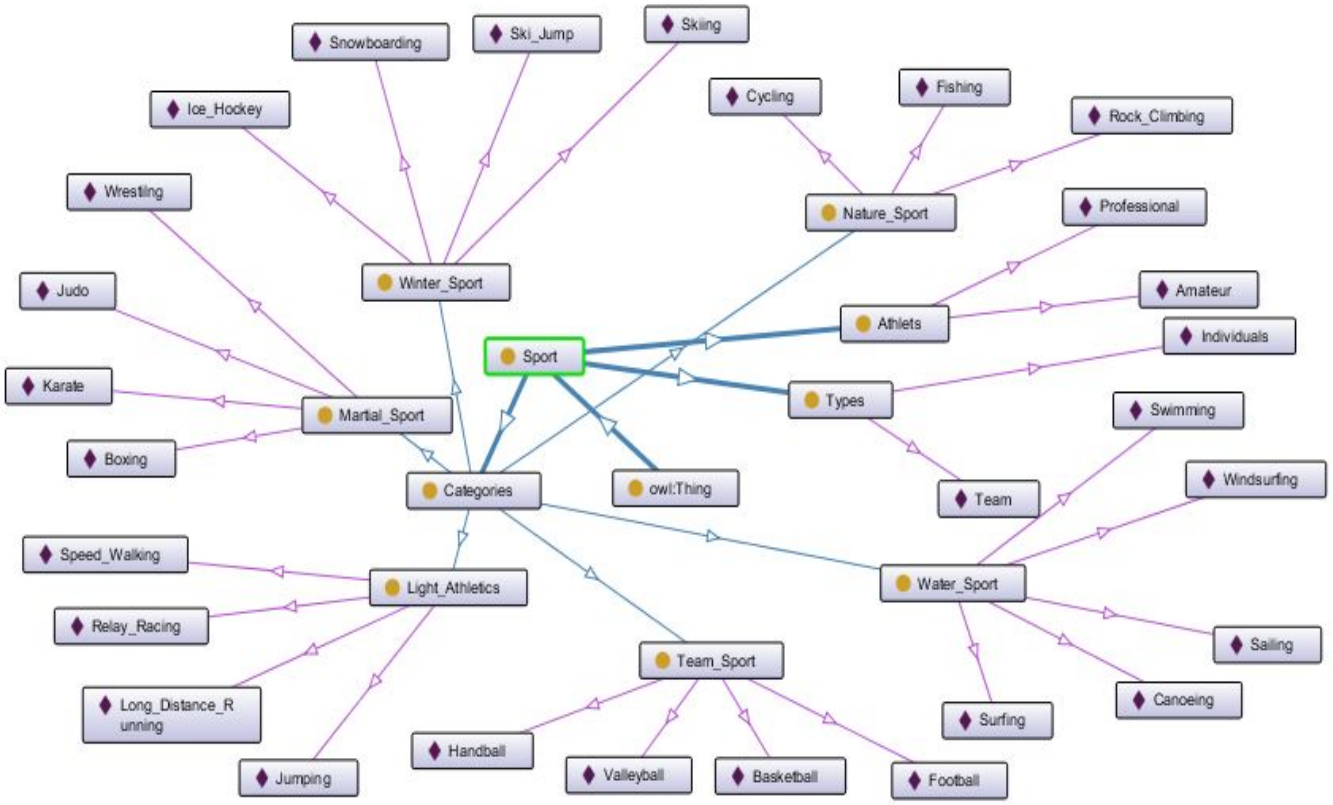
Nelle prossime immagini definiremo le classi e le sottoclassi.

Lo sviluppo dell'**Ontologia** è proseguito con una modalità **mista**, definendo prima i concetti base per poi **generalizzarli** o **specializzarli.** Le proprietà descrivono gli attributi delle istanze della classe. Esse possono essere intrinseche ed estrinseche, semplici e composte. Su esse possono essere applicati dei vincoli riguardanti per esempio la **cardinalità** e il **tipo**.



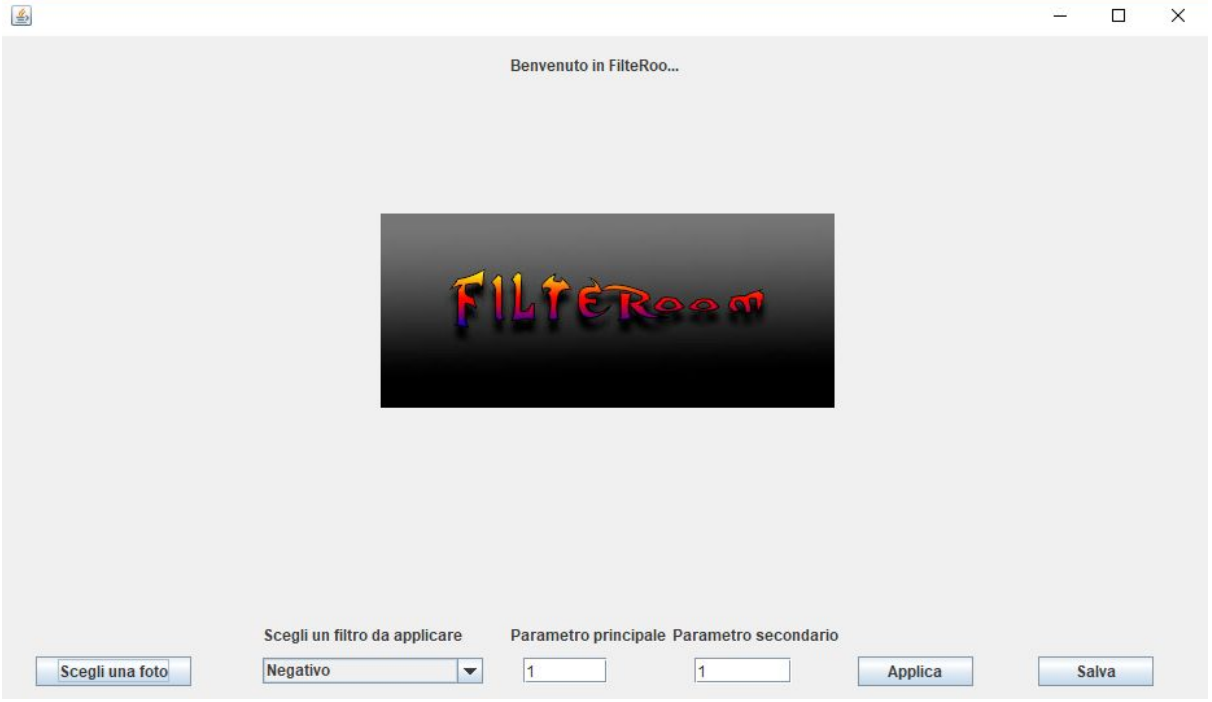
Definiamo ora le istanze della classe

Infine abbiamo creato un grafico dell' **Ontologia.**



OpenCV

OpenCV è una libreria open source che fornisce funzionalità utili nell'ambito della computer vision. Abbiamo realizzato in linguaggio Java un'applicazione che utilizza OpenCV per applicare ad una foto scelta dall'utente una serie di filtri spaziali e il filtro Laplaciano.
I filtri spaziali sono stati realizzati da noi, quello Laplaciano è fornito direttamente nella libreria OpenCV.
Il sorgente dell'applicazione è disponibile al seguente link.



L'applicazione consiste di una semplice interfaccia dalla quale è possibile scegliere una foto dal proprio computer, o da un supporto di memorizzazione esterno inserito in quest'ultimo, selezionare il filtro da applicare e visualizzare sulla sinistra la foto originale e sulla destra la foto modificata.
Inoltre mediante il pulsante salva la foto viene salvata affianco alla foto originale.
Oltre alla selezione del filtro da applicare è possibile scegliere fino a due parametri da applicare al filtro, se il filtro ne ha bisogno.
L'applicazione è stata progettata con l'architettura **View - Controller** in linguaggio java.
Le classi sono due, la classe **View** che gestisce l'interfaccia utente e la classe **Controller** che si occupa delle elaborazioni.
Per lo sviluppo dell'applicazione abbiamo utilizzato la classe Imgproc, inclusa nella libreria OpenCV.
Questa classe fornisce i metodi per le principali elaborazioni possibili sulle immagini.

```
Mat frame = Imgcodecs.imread(path);
```

Il metodo imread, in particolare, consente di leggere un'immagine passando il path della stessa.
L'immagine letta viene salvata in una matrice, a tal proposito è stata utilizzata la classe Mat, anch'essa fornita dalla libreria OpenCV.
La matrice *frame* contiene quindi per ogni elemento il valore di ogni pixel della foto, applicare un filtro ad una foto vuol dire compiere quindi operazioni sulla matrice e in particolare sul valore delle intensità dei pixel.
Se un'immagine è in scala di grigi avrà un solo canale, quindi per ogni pixel vi è un'indicazione del livello di grigio.
Se invece l'immagine è a colori vi sono 3 colori principali, RGB e l'immagine sarà formata da tre canali, ognuno contenenti il livello del colore corrispondente.
I filtri applicano le trasformazioni sia a immagini in bianco e nero, agendo in questo caso sul solo livello di grigio, ma anche alle immagini a colori, agendo su tutti e tre i canali.

Dalla matrice viene poi generata una nuova immagine da poter mostrare sul label dell'interfaccia mediante la seguente funzione.

```
public BufferedImage bufferedImage(Mat m) {
          int type = BufferedImage.TYPE_BYTE_GRAY; //controllo i canali
          if ( m.channels() > 1 ) {
              type = BufferedImage.TYPE_3BYTE_BGR;
          }
          BufferedImage image = new BufferedImage(m.cols(),m.rows(),
type);

m.get(0,0,((DataBufferByte)image.getRaster().getDataBuffer()).getData()
); //restituisce tutti i pixel
          return image;
}
```

I filtri da noi realizzati sono:

- Filtro negativo
- Logaritmico
- Filtro potenza
- Filtro luminosità
- Filtro contrasto
- Filtro gamma

Filtro Negativo

Questo filtro modifica il valore di ogni pixel della matrice e impostando il valore complementare.
In un' immagine in bianco e nero i colori si invertono, se l'immagine è a colori ad ogni colore viene sostituito il suo complementare:

```
if(filtro.equals("Negativo")) { //filtro negativo
                  System.out.println("Applico il filtro negativo");
                  for(int i=0;i<img.rows();i++) {
                        for(int j=0;j<img.cols();j++) {
                              double[] buff = img.get(i, j);
                              for(int h=0;h<buff.length;h++) {
                                    buff[h] = 255.0 - buff[h];
                              }
                              frame.put(i, j, buff);
                        }
                  }
```

```
}
```

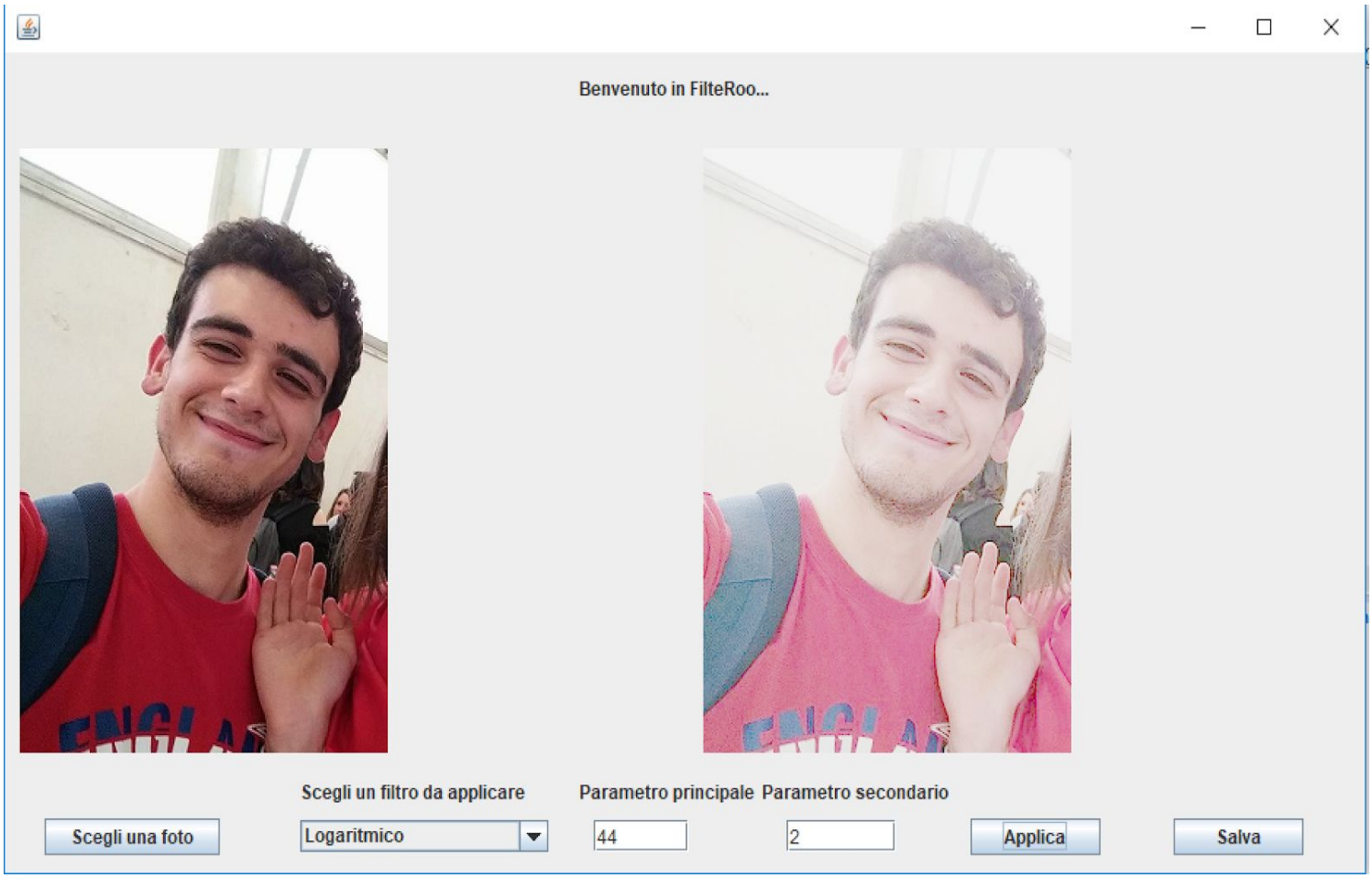
Questo filtro non ha bisogno di parametri aggiuntivi.

Filtro Logaritmo

Questo filtro applica ad ogni pixel la seguente funzione: $c * log(1 + x)$ dove c è una costante, il cui valore è settabile tramite l'interfaccia (la casella a sinistra) e x è il valore precedente del pixel.
Questo filtro schiarisce i colori scuri e lascia quasi invariati i colori chiari.

```
if(filtro.equals("Logaritmico")) { //filtro logaritmico
                  System.out.println("Applico il filtro logaritmico");
                  //converto c in double
                  double cc = Double.parseDouble(c);
                  for(int i=0;i<img.rows();i++) {
                        for(int j=0;j<img.cols();j++) {
                              double[] buff = img.get(i, j);
                              for(int h=0;h<buff.length;h++) {
                                    buff[h]=cc*Math.log(1+buff[h]);
//controllo che venga rispettato il fondo scala
                                    if(buff[h]>255) buff[h] = 255;
                              }
                              frame.put(i, j, buff);
                        }
                  }
}
```

Con questo filtro i valori colorati non vengono amplificati di molto mentre i valori scuri vengono schiariti, il risultato è un'immagine molto più chiara.



Filtro Potenza

Il filtro potenza invece eleva a potenza il valore dei pixel dell'immagine,
è possibile utilizzarlo per valori di potenza maggiori di 1 o per valori di potenza minori di 1.
Per questo filtro è possibile settare entrambi i parametri, infatti ad ogni pixel viene applicata la funzione $p = c * p^g$ dove p è il valore del pixel, c'è una costante che è possibile passare tramite l'interfaccia e g è il valore della potenza (gamma).

```
if(filtro.equals("Potenza")) {//filtro potenza
                  System.out.println("Applico il filtro potenza");
                  //converto c e var in double
                  double cc = Double.parseDouble(c);
                  double gamma = Double.parseDouble(var);
                  for(int i=0;i<img.rows();i++) {
                        for(int j=0;j<img.cols();j++) {
                              double[] buff = img.get(i, j);
                              for(int h=0;h<buff.length;h++) {
                                    buff[h]=cc*Math.pow(buff[h], gamma);
                                    if(buff[h]>255) buff[h] = 255;
```

```
                            }
                            frame.put(i, j, buff);
                        }
                    }
}
```

In caso di valore di gamma maggiore di 1 i colori vengono schiariti.

Al contrario invece i colori vengono tendono verso il nero.

Luminosità

Per aumentare o diminuire la luminosità si aumenta linearmente il valore di luminosità dei pixel, il filtro agisce in questo caso sommando il valore della casella di testo a destra al valore del pixel fino alla saturazione.

```
    if(filtro.equals("Brightness")) { //filtro per correzione luminosità'
```

```
                System.out.println("Applico il filtro luminosità");
                //converto var in double
                double g = Double.parseDouble(var);
                for(int i=0;i<img.rows();i++) {
                      for(int j=0;j<img.cols();j++) {
                            double[] buff = img.get(i, j);
                            for(int h=0;h<buff.length;h++) {
                                  buff[h]=buff[h] + g;
                                  if(buff[h]>255) buff[h] = 255;
                                  if(buff[h]<0) buff[h] = 0;
                            }
                            frame.put(i, j, buff);
                      }
                }
```

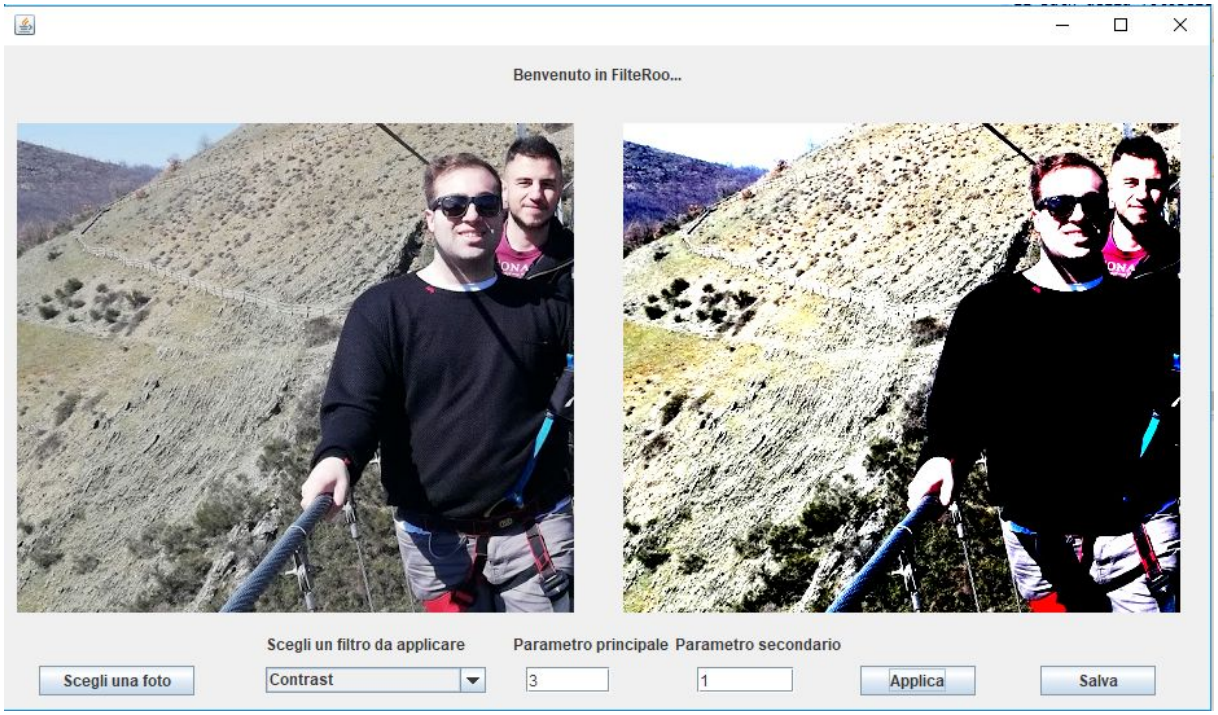
Se il valore della variabile è minore di zero la luminosità viene diminuita.

Contrasto

Il filtro agisce sul valore del contrasto dell'immagine applicando ai pixel l'equazione
$p = m(p - 127) + 127$

```
if(filtro.equals("Contrast")) {//modifico il contrasto
                System.out.println("Applico il filtro contrasto");
                //converto var in double
                double m = Double.parseDouble(var);
                for(int i=0;i<img.rows();i++) {
                      for(int j=0;j<img.cols();j++) {
                            double[] buff = img.get(i, j);
                            for(int h=0;h<buff.length;h++) {
                                  buff[h]= m*(buff[h]-127) +127;
                                  if(buff[h]>255) buff[h] = 255;
                                  if(buff[h]<0) buff[h] = 0;
                            }
                            frame.put(i, j, buff);
                      }
                }
          }
```

Se il valore passato dall'interfaccia è un valore maggiore di 1 il contrasto viene aumentato, altrimenti se il valore è compreso da 0 e 1 il valore viene diminuito.
I colori medi restano invariati, il filtro nel primo caso tende a saturare i colori, ovvero i colori chiari diventano ancora più chiari e quelli scuri ancora più scuri.
Nel secondo caso i colori tendono tutti al valor medio, in questo caso il grigio.



Ecco un esempio di applicazione del filtro contrasto con parametro maggiore di uno, quindi il contrasto viene aumentato. Si nota subito che i colori tendono a diventare più accesi.

Gamma

Il filtro gamma modifica la gamma dei colori saturando stavolta i colori medi, i colori tendenti al bianco e al nero restano invariati.
Il filtro applica l'equazione $p = 255 * [p/255]^{1/g}$ dove p è il valore del pixel, e gamma è un parametro passato tramite interfaccia.
Il valore passato al programma può essere maggiore di uno o compreso tra zero ed uno.

Nel secondo caso i colori medi vengono schiariti, i colori tendenti al bianco o al nero restano comunque invariati.

Il filtro è stato realizzato mediante il seguente codice:

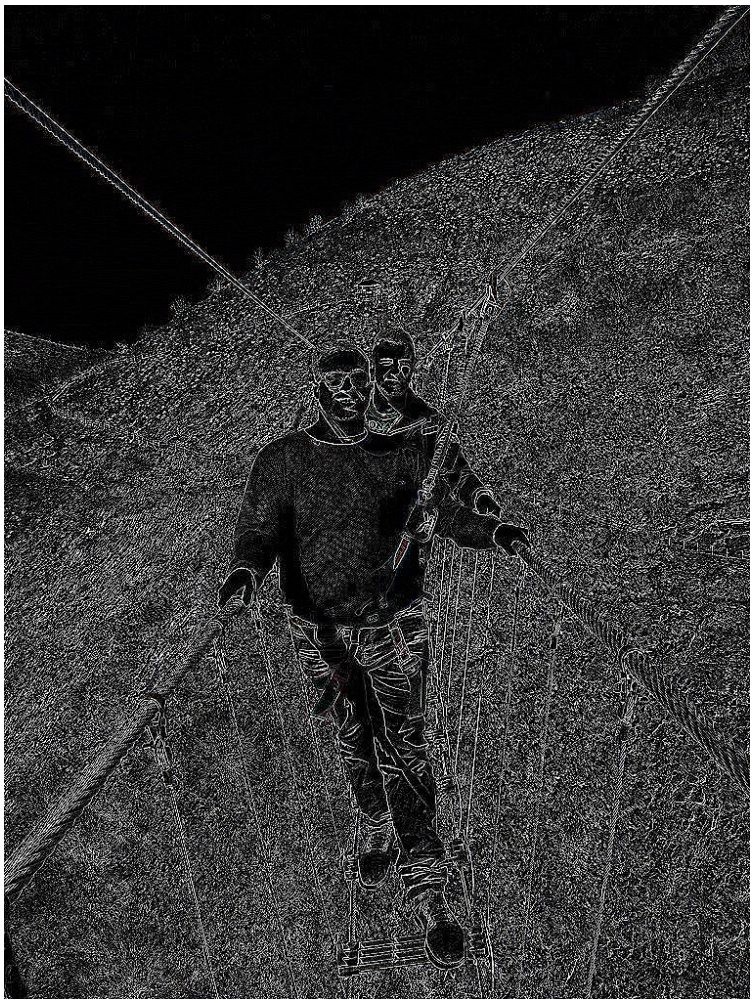
```
if(filtro.equals("Gamma")) {//modifico la gamma
                  System.out.println("Applico il filtro gamma");
                  //converto var in double
                  double gamma = Double.parseDouble(var);
                  for(int i=0;i<img.rows();i++) {
                        for(int j=0;j<img.cols();j++) {
                              double[] buff = img.get(i, j);
                              for(int h=0;h<buff.length;h++) {
                                    double buf = buff[h];
                                    buf = buf/255.0;
                              buff[h]= 255.0*(Math.pow(buf, 1/gamma));
                                    if(buff[h]>255.0) buff[h] = 255.0;
                                    if(buff[h]<0) buff[h] = 0;
                              }
                              frame.put(i, j, buff);
                        }
                  }
            }
```

Filtro Laplaciano

abbiamo inoltre realizzato il filtro laplaciano utilizzando la funzione fornita dalla classe Imgproc di OpenCV.

```
if(filtro.equals("Laplaciano")) {
                  System.out.println("Applico il filtro laplaciano");
                  //converto i parametri
                  double scale = Integer.parseInt(c);
                  int ksize = Integer.parseInt(var);
Imgproc.Laplacian(img, frame, img.depth(), ksize, scale);
            }


            return frame;
      }
```

Di seguito ecco un esempio di applicazione di un filtro laplaciano



Per il salvataggio della foto abbiamo utilizzato la funzione imwrite fornita da OpenCV modificando il nome della foto, conservando però il path.
La foto quindi viene salvata affianco alla foto scelta con estensione jpg.

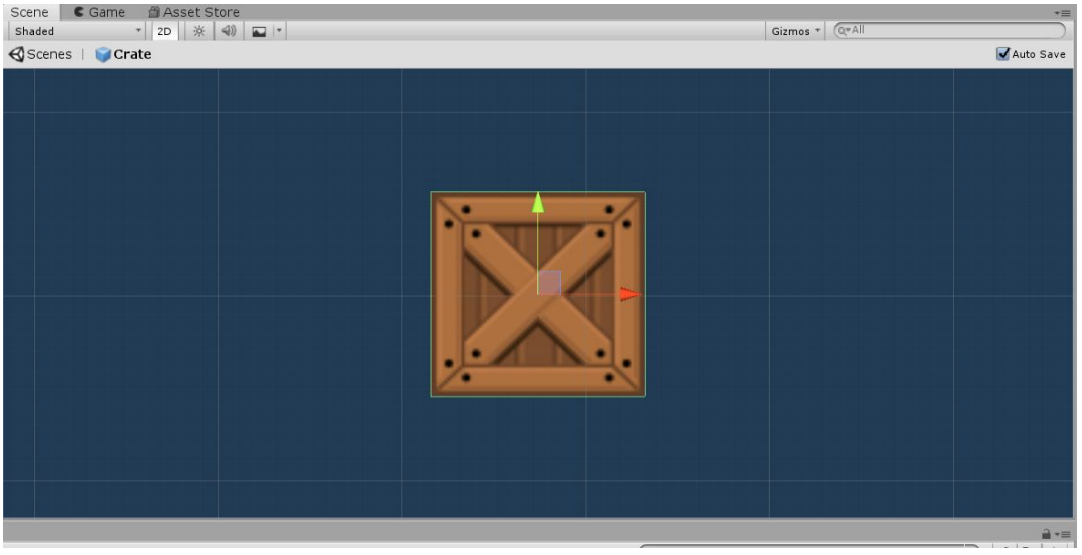
```
Imgcodecs.imwrite(path,this.frame);
```

Unity (Pac-Form)

Unity è un applicativo d'autore integrato multipiattaforma per la creazione di videogiochi 3D/2D o altri contenuti interattivi, quali visualizzazioni architettoniche o animazioni 3D in tempo reale.
L'ambiente di sviluppo Unity gira sia su Microsoft Windows sia su macOS, e i giochi che produce possono essere eseguiti su Microsoft Windows, Mac, Linux, Xbox 360, PlayStation 3, PlayStation Vita, Wii, iPad, iPhone, Android, Windows Mobile e, nel corso dell'anno 2014, PlayStation 4, Xbox One e Wii U.

Io e il mio team abbiamo deciso di implementare un gioco che vede come protagonista pacman in una veste totalmente nuova. Abbiamo deciso di inserirlo in un ambiente platform cercando di dare nuova vita ad un titolo che ha fatto la storia del gaming.
Abbiamo creato una mappa totalmente da zero e inserendo elementi che cercano di coinvolgere il video giocatore.
Qui faremo una descrizione di tutti gli elementi inseriti nel livello:

Elementi di Gioco:

- Crate: 

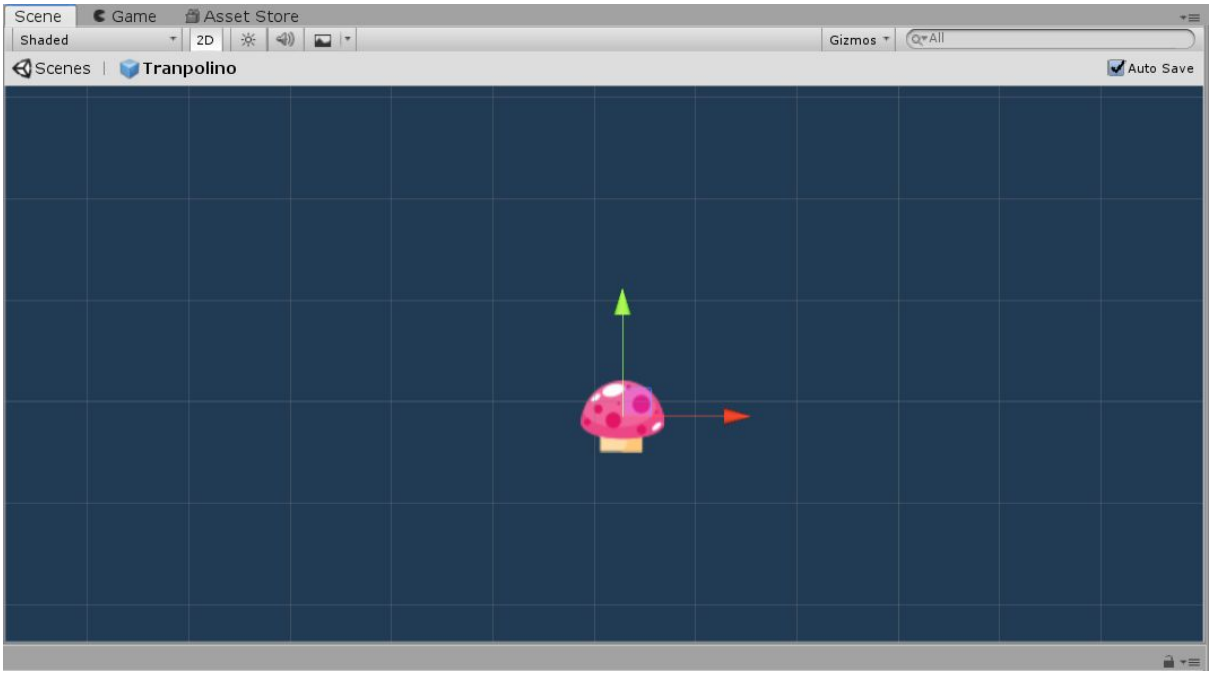
  le caratteristiche del crate sono:

Ogni Crate ha oltre allo sprite, un rigidbody 2d in modo da essere soggetto a fisica, un box collider 2D ed uno Script.
Lo script è molto semplice, serve a ricostruire l'oggetto Crate in caso venisse distrutto.

```
using System.Collections;
```

```
using System.Collections.Generic;
using UnityEngine;

public class Fantasmi2 : MonoBehaviour
{
    public Transform S; //sinistra
    public Transform D;//destra
    public float V;//velocità
    public Rigidbody2D rb;
    private bool Mov=false;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if (Mov && transform.position.x > D.position.x) //se ci stiamo muovendo
verso destra e ha raggiunto la posizione destra
        {
            transform.localScale = new Vector3(-0.4832942f, 0.4832942f,
0.4832942f);//ruota lo sprites del fantasma
            Mov = false;
        }

        if (!Mov && transform.position.x < S.position.x) //se ci stiamo muovendo
verso sinistra e ha raggiunto la posizione sinistra
        {
            transform.localScale = new Vector3(0.4832942f, 0.4832942f,
0.4832942f);//ruota lo sprites del fantasma
            Mov = true;
        }

        if (Mov)
        {
            rb.velocity = new Vector3(V, rb.velocity.y, 0f);// applica una forza
verso destra
        }
        else { rb.velocity = new Vector3(-V, rb.velocity.y, 0f); } //applica una
forza verso sinistra

    }
}
```

-Trampolino:



Caratteristiche di ogni Trampolino:
Le caratteristiche del trampolino sono due polygon collider in cui uno ha la funzione is trigger applicato. Quel collider serve per attivare la funzione OnColliderEnter 2D all'interno della funzione Trampolino.cs:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Trampolino : MonoBehaviour
{
    public float Spinta; //aggiunta al salto
    public Movement movimento; //oggetto di tipo movement
    // Start is called before the first frame update
    void Start()
    {
       // movimento = FindObjectsOfType<Movement> ();
    }

    // Update is called once per frame
    void Update()
    {

    }
    void OnTriggerEnter2D(Collider2D other)//funzione  che alla collisione con un
```

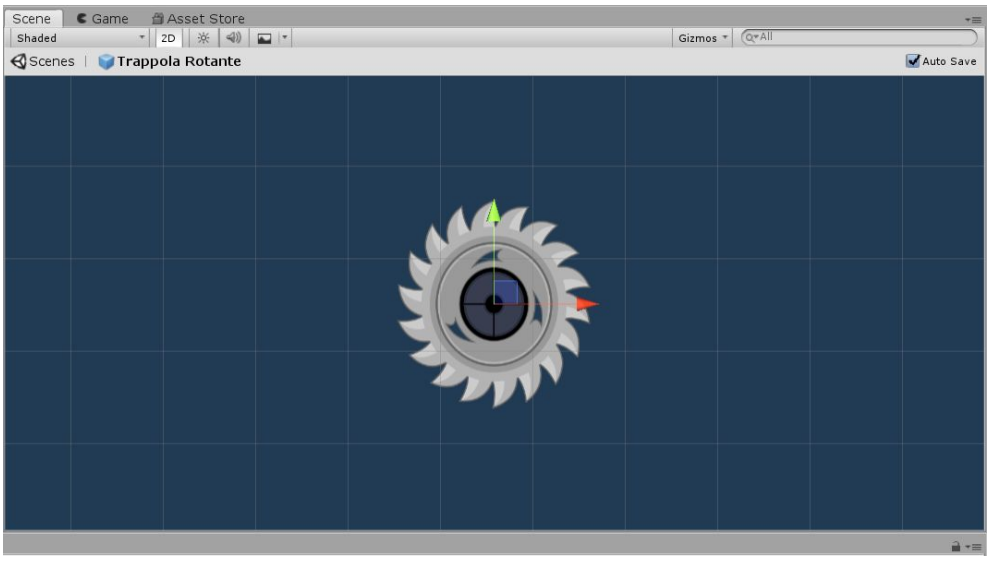
```
collider che ha la spunta a istrigger entra nella funzione
    {
        if (other.tag == "Player")//controlla se il tag con cui è colliso è di
tipo player
        {
            movimento.rb.AddForce(transform.up * Spinta);// aggiunge al rigidbody
una forza
        }
    }
}
```

-Moneta:

Caratteristiche:
Le caratteristiche della moneta sono un circle collider e uno script:

```
    {
        if (other.tag == "Player")
        {
            GMonete.Coins++;// Serve per accedere a gesyione monete e aumentare
il punteggio delle monete raccolte
            Destroy(gameObject);//distrugge la moneta
        }
    }
}
```

-Trappola Rotante:



La Trappola rotante ha due script che regolano il suo comportamento:
Script Movimento oggetti:

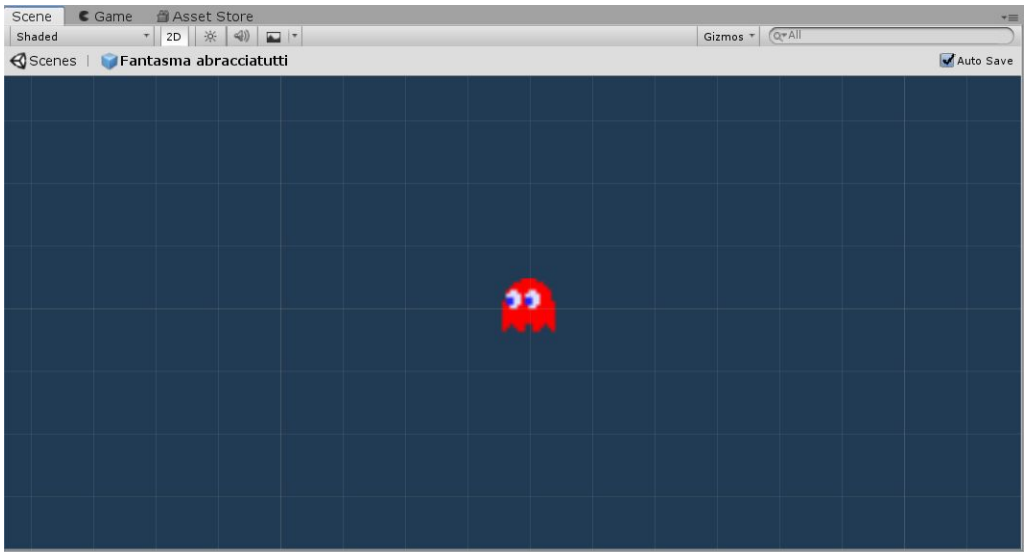
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MovimentoOggetti : MonoBehaviour
{
    public GameObject OggettoDaSpostare; // definisco un oggetto
    public Transform PuntoIniziale; //definisco un transform che ci servirà come
punto iniziale
    public Transform PuntoFinale;//definisco un transform che ci servirà come
punto finale
    public float Velocita; //velocità della nostra piattafomra
    private Vector3 Target; // sarà il prossimo obiettivo per la nostra
piattaforma, se abbiamo toccato l'obiettivo iniziale il
                            // prossimo obbiettivo sarà il punto finale e così
via...

    // Start is called before the first frame update
    void Start()
    {
        Target = PuntoFinale.position;
    }

    // Update is called once per frame
    void Update()
    {
        OggettoDaSpostare.transform.position =
Vector3.MoveTowards(OggettoDaSpostare.transform.position, Target, Velocita *
Time.deltaTime); //
```

```
        if (OggettoDaSpostare.transform.position == PuntoFinale.position) // se
la piattaforma è arrivata la pt finale

//target diventa la posizione del pt iniziale
        {
            Target = PuntoIniziale.position;
        }
        if (OggettoDaSpostare.transform.position ==PuntoIniziale.position) // se
la piattaforma è arrivata la pt iniziale

//target diventa la posizione del pt finale
        {
            Target = PuntoFinale.position;
        }
    }
}
```

Script Danni:

```
    void OnTriggerEnter2D(Collider2D other) //funzione  che alla collisione con
un collider che ha la spunta a istrigger entra nella funzione
        {
            if (other.tag == "Player")//controlla se il tag con cui è colliso è
di tipo player
            {
            levelManagment.Danneggiare(DanniDaDare);// stabilire i danni di ogni
elemento
            }
        }


}
```

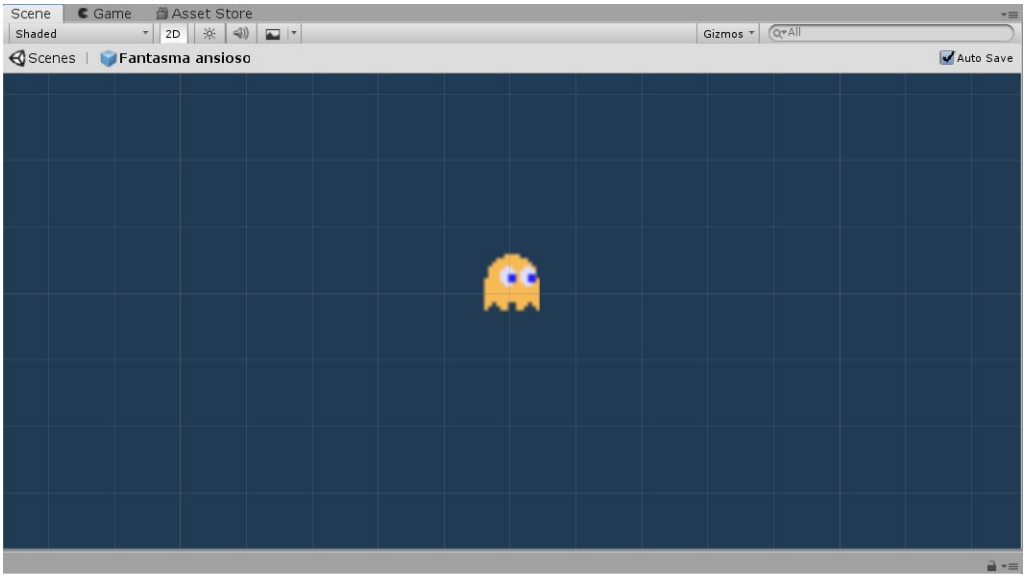
Fantasma Tipo 1:
Il fantasma 1 si compone di due script, uno danni e l'altro fantasma move.



Fantasma Muve:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FantasmaMuve : MonoBehaviour
{
    public float velocita;
    private bool movimento=true; //stabilisce se ci sta un movimento
    public Rigidbody2D rb;
    // Start is called before the first frame update
    void Start()
    {
        rb = GetComponent<Rigidbody2D>(); //associa al movimento un componente
rigidbody
    }

    // Update is called once per frame
    void Update()
    {
        if (movimento)
        {
            rb.velocity = new Vector3(-velocita, rb.velocity.y, 0f);//applica una
forza al fantasma

        }
    }
    void attivazione() //setta movimento a true
    {
        movimento = true;
    }
```

```
    void OnTriggerEnter2D(Collider2D other)//quando il fantasma cade e quindi
tocca il destroyer oppure destroyer1 viene distrutto
    {
        if (other.tag == "Destroyer" || other.tag=="Destroyer 1")
        {
            Destroy(gameObject);
        }
    }

}
```



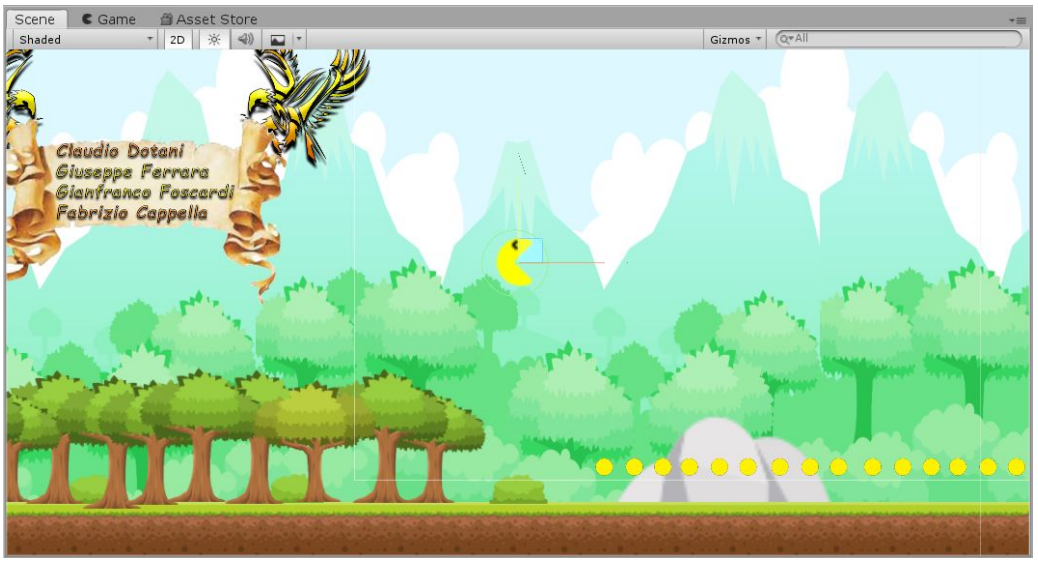
Fantasma Tipo 2:
Come quello sopra citato, questo si compone di 2 script, uno per il movimento e uno per i danni, ora riportiamo solo quello per il movimento perchè quello per i danni è uguale.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Fantasmi2 : MonoBehaviour
```

```
{
    public Transform S; //sinistra
    public Transform D;//destra
    public float V;//velocità
    public Rigidbody2D rb;
    private bool Mov=false;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if (Mov && transform.position.x > D.position.x) //se ci stiamo muovendo
verso destra e ha raggiunto la posizione destra
        {
            transform.localScale = new Vector3(-0.4832942f, 0.4832942f,
0.4832942f);//ruota lo sprites del fantasma
            Mov = false;
        }

        if (!Mov && transform.position.x < S.position.x) //se ci stiamo muovendo
verso sinistra e ha raggiunto la posizione sinistra
        {
            transform.localScale = new Vector3(0.4832942f, 0.4832942f,
0.4832942f);//ruota lo sprites del fantasma
            Mov = true;
        }

        if (Mov)
        {
            rb.velocity = new Vector3(V, rb.velocity.y, 0f);// applica una forza
verso destra
        }
        else { rb.velocity = new Vector3(-V, rb.velocity.y, 0f); } //applica una
forza verso sinistra

     }
}
```

-PacMan (Hero):

Ha un rigid body 2D che ha lo script: movement




```
      }
    void OnTriggerEnter2D(Collider2D other) // funzione del respawn
    {
        if (other.tag == "Destroyer") // quando entra in collisione con un
oggetto di tipo destroyer entra nel ciclo
        {
            Levelmanager.hp = 0; // qunado moriamo deve settare gli hp a 0
            Levelmanager.HeartSlide.value = Levelmanager.hp; // passa alla slider
il valore degli hp
            Levelmanager.Respawn(); //va alla funzione di spawn
        }
        if (other.tag == "CheckPoint")// quando entra in collisione con un
oggetto di tipo checkpoint entra nel ciclo
        {
            SpawnPosition = other.transform.position;
        }
    }
    private void OnCollisionEnter2D(Collision2D other) // funzione che si attiva
quando hero entra in contatto con con collider
    {
        if (other.gameObject.tag == "Piattaforma")
        {
            transform.parent = other.transform;//serve per congelare la posizione
sulla piattaforma
        }
    }
    private void OnCollisionExit2D(Collision2D other)// quando esce dalla
collisione si attiva la funzione
    {
        if (other.gameObject.tag == "Piattaforma")
```

```
        {
            transform.parent = null;//scongela hero
        }
    }


}
```

LEVEL MANAGEMENT

Questo elemento gestisce la vita del player e il respawn.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class LevelManagment : MonoBehaviour
{
    public float TempoDattesa;//tempo di attesa prima di ricreare il giocatore
    public Movement Movimento; // movement fa riferimento allo script movement.cs
    public int hp=10;
    public int MaxHp;
    public Slider HeartSlide; // fa riferimento ad un oggetto di tipo Slider
(barra della vita)
    private bool rinascita;
    // Start is called before the first frame update
    void Start()
    {
        Movimento = FindObjectOfType<Movement>();// Nel momento in cui il gioco
inizia il nostro movimento è uguale alla ricerca
                                                 //dell'oggetto movement cioè
movement è associato ad hero, quindi fa riferimento a lui
        hp = MaxHp;
        HeartSlide.value = hp;
    }

    // Update is called once per frame
    void Update()
    {
        HeartSlide.value = hp;
        if (hp <= 0 && !rinascita) // se i nostri hp sono negativi o uguale a
zero e rinascita è falsa
        {
            Respawn();
```

```
            rinascita = true; //serve per fargli fare un solo respawning
        }
    }

    public void Respawn()// funzione che ricostruisce il player
    {
        StartCoroutine(Spawntime());
    }

    IEnumerator Spawntime() //funzione che permette di ricostruire il giocatore
dopo un certo tempo
    {
        Movimento.gameObject.SetActive(false);// settiamo l'attività del player a
falso quindi inattiva
        yield return new WaitForSeconds(TempoDattesa); //permette di eseguire un
conto alla rovescia dopo il quale viene ricostruito il giocatore
        hp = MaxHp; //setta gli hp al valore di hp massimi
        rinascita = false;
        Movimento.transform.position = Movimento.SpawnPosition; //la posizione
dell'oggetto movimento è uguale alla posizione settata in movement
        Movimento.gameObject.SetActive(true);//riattiviamo l'oggetto
    }
    public void Danneggiare (int Danni)// funzione che crea danni
    {
        hp -= Danni;
    }
}
```

Camera

Per gestire la camera abbiamo deciso di ancorarla al Player, in modo tale che quest'ultimo stia sempre al centro della scena.

```
using UnityEngine;
using System.Collections;

public class Camera : MonoBehaviour {
      public Movement player;
      public bool seguendo;
      public float xOffset;
      public float yOffset;

      // Use this for initialization
      void Start () {
            player = FindObjectOfType<Movement> ();
```

```
            seguendo= true;
      }

      // Update is called once per frame
      void Update () {
            if(seguendo)
                  transform.position = new Vector3(player.transform.position.x +
xOffset,player.transform.position.y + yOffset ,transform.position.z);
      }
}
```

Conclusioni

Fino a un decennio fa, era impensabile che una macchina riuscisse a comprendere il linguaggio umano, che per natura è ricco di ambiguità (o almeno riusciva a interpretarlo solo in ambiti ben definiti). Ad oggi (2019) invece per mezzo di **Google Duplex (2018),** è possibile interloquire con una macchina come se fosse un umano. Di seguito è riportato un breve video di pochi secondi che mostra **Duplex** in azione (In questo esempio vediamo che l'assistente vocale prenota per conto di terzi, un appuntamento dal parrucchiere).