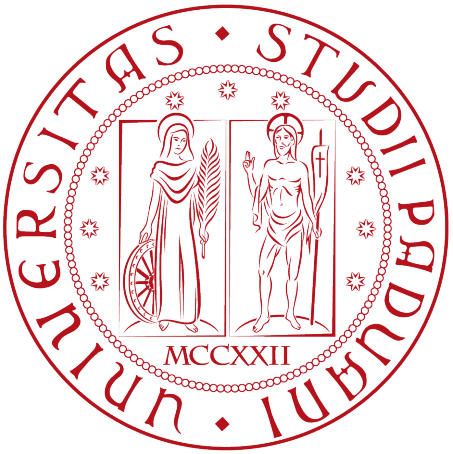
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Analisi e sviluppo di un'applicazione per la configurazione automatica di una chatbot professionale

Tesi di laurea triennale

Relatore
Prof.Ballan Lamberto

Laureando
Stefano Zanatta

Anno Accademico 2018-2019

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di trecentododici ore, dal laureando Stefano Zanatta presso l'azienda PAT s.r.l. Gli obbiettivi principali del progetto erano:

* studio del motore semantico di Engagent, la chatbot professionale dell'azienda;
* studio dei risultati di un algoritmo di clustering, sviluppato da ricercatori esterni all'azienda;
* integrazione automatica dei cluster con Engagent, eseguendo operazioni di pos-tagging.

Ringraziamenti

Innanzitutto, vorrei ringrazione il Prof. Lamberto Ballan, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura della tesi.
Vorrei ringraziare il tutor aziendale Davide Bastianetto, per avermi dato l'opportunità di svolgere lo stage alla PAT.
Desidero ringraziare con affetto i miei genitori e mio fratello per il sostegno morale (ed economico) dimostratomi durante questi anni.

Padova, Settembre 2019 Stefano Zanatta

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 L'azienda

PAT s.r.lè un'azienda italiana che da 25 anni sviluppa soluzioni software per altre aziende e privati.
L'azienda lavora su 5 diversi macro-progetti, uno dei quali è Engagent, la chatbot professionale oggetto dei miei due mesi di stage.
Dal 2013, PAT è entrata a far parte di Zucchetti Group.

Figura 1.1: logo PAT s.r.l

1.1.1 Prodotti e servizi

L'azienda offre ai suoi clienti l'automatizzazione dei processi e il miglioramento dell'*user experience* dei loro prodotti. PAT concretizza quesi obbiettivi attraverso i seguenti prodotti:

* Engagent: chatbot semi-automatizzata per uso professionale, l'unico prodotto di PAT s.r.lcon cui sono entrato in cotatto;
* Helpdesk;
* Infinite: *CRM* software orientato alla relazione tra cliente e azienda;
* Brain *Interactive*: piattaforma per governare dei servizi personalizzati attraverso dei diagrammi di flusso;

* Teammee: piattaforma per la comunicazione tra i dipendenti in un'azienda, usando la logica dei social networks;

Engagent

Engagent è una chatbot orientata al business, con un agente virtuale integrato. In *backend*, un motore semantico permette di capire cosa sta chiedendo l'utente e trovare la risposta più coerente. Se la domanda è troppo complessa, il motore semantico estrate la categoria della domanda e la reinderizza all'operatore adeguato.

Figura 1.2: logo Engagent

1.1.2 Organizzazione e Metodo di lavoro

L'azienda è divisa in più gruppi di lavoro, uno per ogni macro-progetto, oltre alla segreteria e direzione.
Ogni team è separato dagli altri, anche se la collaborazione tra le parti è necessaria.
Tutti i team di sviluppo in PAT s.r.lseguono una metodologia Agile. Questa fa parte delle metodologie iterative, caratterizzata da brevi iterazioni (o sprint, di circa 3-4 settimane) seguite dalla *review* del lavoro svolto. Il focus principale si trova nel cliente: ci deve essere una interazione costante per capire quali sono le *feature* più importanti, che hanno precedenza sulle altre.
Il team a cui ho preso parte applica questa metodologia. Il contatto con il cliente è frequente, che sia manutenzione o nuove *features* da sviluppare. La piccola dimensione del team e le riunioni giornaliere permettono una buona collaborazione. Il team è gestito da un responsabile che organizza le riunioni e comunica con il manager dell'azienda.
Per quanto mi rigurarda, ho adottato senza difficoltà queste metodologie, perché molto simili a quelle utilizzate durante il progetto di ingegneria del software.

Figura 1.3: Agile

1.2 Strumenti e Tecnologie

Questa sezione descrive, ad alto livello, le tecnologie utilizzate durante lo stage.

1.2.1 Ambiente di lavoro

L'ambiente di lavoro utilizzato è Windows 10, assieme al pacchetto office per la maggior parte delle attività.
I team di sviluppo hanno libera scelta sugli editor. Per la codifica e la stesura dei docuemnti, ho scelto Visual studio. Per la creazione di diagrammi UML, ho utilizzato Astah UML.
Ogni sviluppatore ha a disposizione un PC fisso e un portatile per le riunioni.

1.2.2 Linguaggi di programmazione

I linguaggi di programazione che ho utilizzato sono:

* **Python:** per sviluppare l'applicazione;
* **Shell:** per la creazione di script per automatizzare alcuni task:
  - esecuzione del programma tramite linea di comando;
  - pulizia della cache del programma;
  - pulizia dei file di output del programma.

1.3 Organizzazione del testo

Il primo capitolo contiene una panoramica dell' azienda e le tecnologie utilizzate PAT s.r.l;

Il secondo capitolo contiene la pianificazione progetto;

Il terzo capitolo approfondisce nel dettaglio i requisiti del progetto, descrivendo il processo di analisi che ha portato alla loro stesura;

Il quarto capitolo approfondisce l'architettura del software sviluppato, con il supporto di esempi specifici e schemi ad alto livello;

Il quinto capitolo contiene il resoconto del progetto e considerazioni personali sullo stage.

Capitolo 2

Descrizione dello stage

2.1 Il progetto

Il progetto è nato dalla necessità dell'azienda PAT s.r.ldi automatizzare il processo di configurazione del motore semantico di *Engagent*[g], il quale richiede la creazione manuale di *synset*[g]e *regole*[g]. Questo processo è economicamente fattibile e vantaggioso finché le dimensioni delle regole create sono ridotte, ma il costo cresce esponenzialmente con l'aumentare delle regole.
Più regole rendono più precisa la chatbot, ma incrementano di conseguenza i [g]synset da inserire, prolungando i tempi di compilazione del'*NLP*[g]da qualche giorno a settimane.
Per risolvere questo problema, PAT s.r.lha scomposto il processo nei seguenti *task* da automatizzare:

1. creazione delle *regole*[g];
2. creazione dei synset;
3. raffinamento dei risultati;
4. creazione del file di configurazione *NLP* per il motore semantico;

2.1.1 Creazione delle regole

Questo task è il più difficile da automatizzare, perché richiede la definizione di *match* contenenti categorie correlate tra loro. Inoltre, non esistono delle regole uguali per tutti, ma ogni settore ha regole diverse.
La soluzione è stata trovata nell'intelligenza artificiale, più in particolare nel clustering. Tramite l'analisi di *chat* e *FAQs* archiviate, è possibile generare delle regole allo stato grezzo.
Il problema è la bassa affidabilità dei risultati. Possibili soluzioni:

- più dati in input (poco realizzabile nel breve periodo);
- algoritmo più complesso (in lavorazione);
- raffinamento manuale dei risultati. Richiede meno tempo di creare l'NLP da zero, soluzione più semplice nel breve termine;

* raffinamento automatico dei risultati (tramite *POS-tagging*). Buon compromesso tra il raffinamento manuale e le altre due soluzioni.

2.1.2 Creazione dei synset

La creazione dei *synset* è facilmente automatizzabile (se le regole sono già state create), in quanto basta trovare i sinonimi delle categorie.

2.1.3 Raffinamento dei risultati

I risultati dell'algoritmo di clustering devono essere ripuliti da *stop-words* e regole prive di significato.

2.1.4 creazione dell'NLP

Adattamento dell'output dell'algoritmo di clustering al motore semantico di Engagent.

2.2 Obbiettivi Aziendali

L'obbiettivo di automatizzare il processo di configurazione di Engagent non è nato con il progetto di stage, ma qualche anno fa, mentre il *Machine Learning* diventava sempre più popolare. L'azienda attribuì questo compito a un team esterno. Il loro compito consisteva nel sviluppare un algoritmo di *ML*, per la generazione di cluster contenenti gli ingredienti essenziali alla configurazione del loro motore semantico.
Verso l'inizio dell'anno, i progressi fatti da questo team erano convincenti, quindi PAT s.r.laveva l'intenzione di sperimentare l'integrazione di tali risultati con il proprio sistema.

2.3 Obbiettivi personali

Durante la ricerca dell'azienda per lo stage, volevo contribuire a un progetto software in ambito professionale, facendo contemporaneamente i primi passi nel mondo delle intelligenze artificiali.
Il progetto proposto da PAT racchiudeva queste prerogative: sarei stato inserito in un progetto maturo e, con l'aiuto di esperti nel settore, avrei potuto lavorare con degli algoritmi di clustering.

2.4 Pianificazione

Con l'aiuto del tutor aziendale, ho redatto il piano di lavoro, che comprende 312 ore distribuite in 8 ore al giorno, per 5 giorni alla settimana (lunedì 24 luglio mi sono dovuto assentare da lavoro, con il consenso del tutor aziendale, per un esame universitario).
La pianificazione ha avuto delle modifiche durante l'avanzare del progetto, vista la sua natura "sperimentale". Per esempio, il linguaggio di programmazione Python è stato accordato assieme al tutor aziendale solamente dopo un'analisi approfondita del problema. Di seguito viene riportata l'ultima versione del piano di lavoro.

* **I settimana:** studio della piattaforma *Engagent*;

* **II settimana:** analisi e stesura del report riguradante il problema descritto in 2.1;
* **III settimana:** ricerca e sperimentazione di possibili soluzioni già esistenti per automatizzare la generazione di sinonimi; analisi e progettazione dell' applicazione NLP-Generator;
* **IV settimana:** preparazione dell'ambiente di lavoro; codifica di NLP-Generator; stesura di test di unità;
* **V settimana:** codifica e miglioramento delle prestazioni di *NLP-Generator*; verfica dei risultati di *NLP-generator* da parte del tutor aziendale
* **VI settimana:** analisi sul miglioramento dei risultati di *NLP-Generator* e codifica; documentazione;
* **VII settimana:** documentazione e validazione;
* **VIII settimana:** collaudo.

Capitolo 3

Analisi dei requisiti

3.1 Casi d'uso

La progettazione dell'applicazione è iniziata con la stesura dei casi d'uso, supportati da diagrammi dei casi d'uso coerenti con lo standard UML.
I casi d'uso sono aumentati durante tutta la durata dello stage. Durante lo sviluppo, assieme al tutor, venivano individuate nuove funzionalità del programma per migliroare i risultati e aumentare l'automazione.

UC0: Scenario principale

Attori Principali: Utente.

Precondizioni: Il sistema è stato installato correttamente. L'utente ha aperto una *shell* posizionata nella root dell'applicazione. (stato principale del sistema).

Descrizione: Il sistema, tramite CLI (*command line interface*), permette di:

* 1 inserire un file json;
* 2 inserire un file xmlsx;
* 3 personalizzare i parametri in output;
* 4 attivare o disattivare le funzionalità del programma;
* 5 creare una configurazione per il motore semantico di Engagent;
* 6 caricare automaticamente l'output in Engagent;
* 7 inserire in input una configurazione già esistente

.

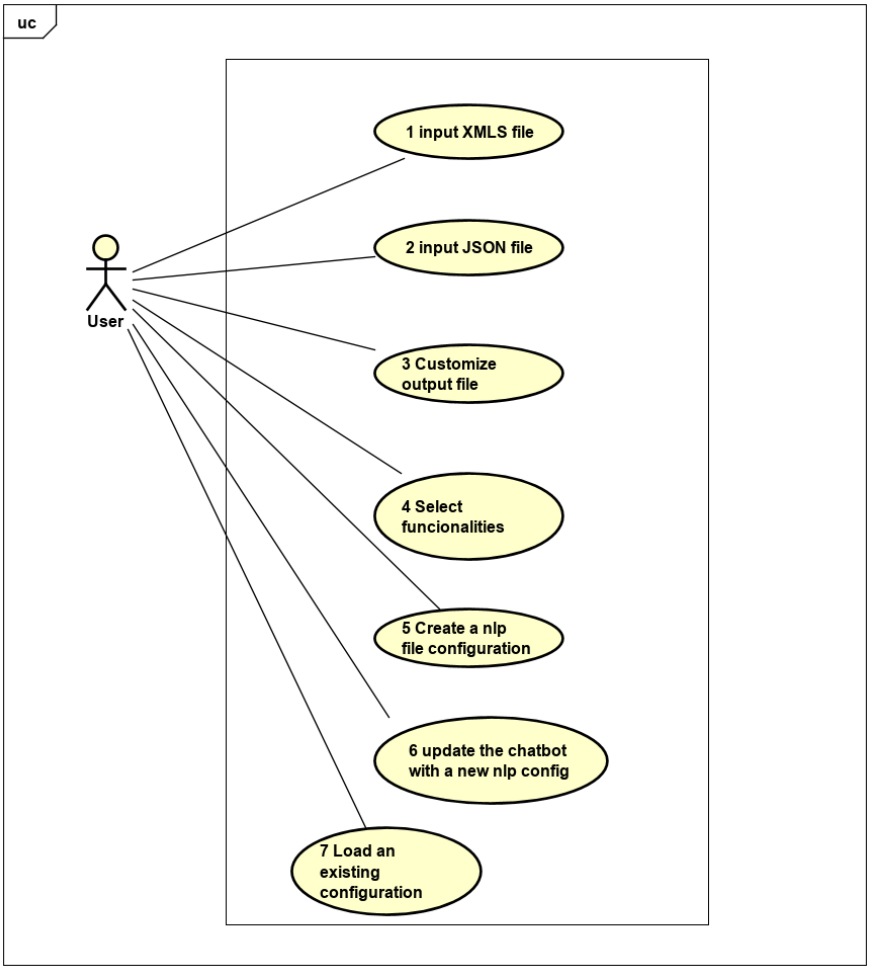


Figura 3.1: Use Case - UC0: Scenario principale

Postcondizioni: Il sistema è pronto per una nuova iterazione.

UC1/2: inserire un file json/xmlsx

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione un comando per l'input di un file json/xmlsx.

Descrizione: L'utente, tramite CLI, inserisce un file json/xmlsx.

Postcondizioni: Il sistema permette di inserire un nuovo file in input.

UC3: Personalizzazione parametri in output

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione dei comandi per la personalizzazione

dell'output.

Descrizione: L'utente, tramite CLI, personalizza i parametri di output:

* 3.1 dominio;
* 3.2 lingua;
* 3.3 prefissi;
* 3.4 priorità delle regole.

.

Postcondizioni: Il sistema torna allo stato principale.

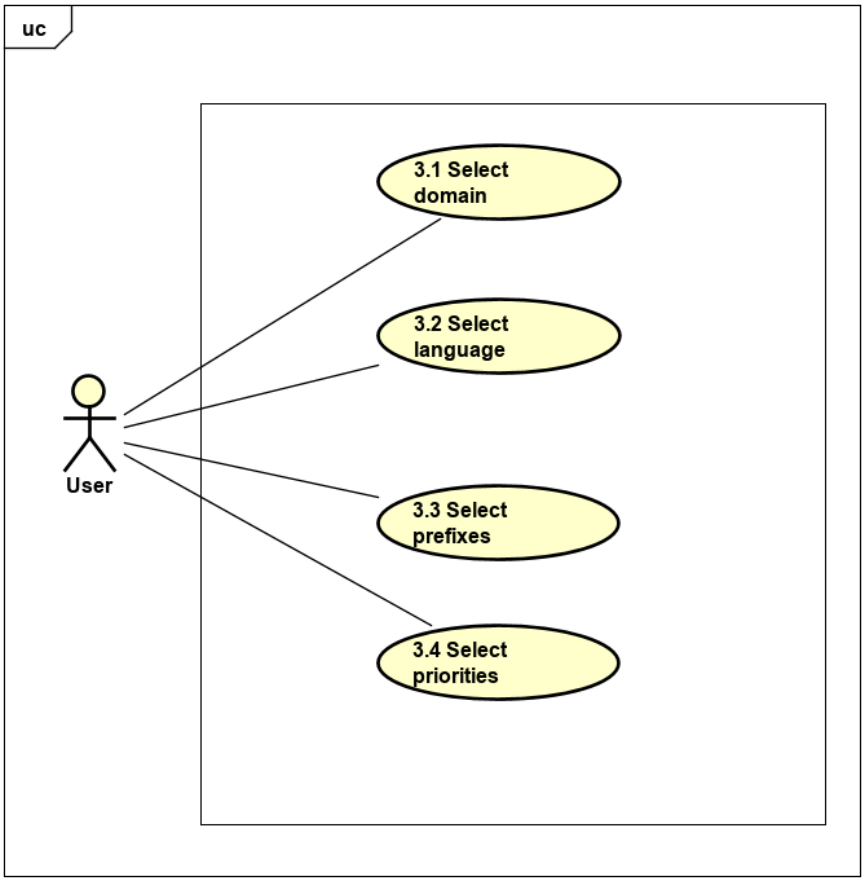


Figura 3.2: Use Case - UC3: Personalizzazione parametri in output

UC3.1, 3.2, 3.3, 3.4: Personalizzazione dominio/lingua/prefissi/priorità delle regole

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione dei comandi per la personalizzazione

dell'output.

Descrizione: L'utente, tramite CLI, modifica il dominio/lingua/prefissi/priorità delle regole.

Postcondizioni: Il sistema permette di personalizzare nuovi parametri.

UC4: Attivare o disattivare le funzionalità del programma

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione dei comandi per selezionare quali funzionalità del programma utilizzare.

Descrizione: L'utente, tramite CLI, attiva o disattiva le seguenti funzionalità:

* 4.1 stemming sulle categorie;
* 4.2 validazione delle categorie attraverso le domande associate;
* 4.3 rimozione delle parole presenti in blacklist;
* 4.4 creazione di *cluster*;
* 4.5 creazione di *intent*.

.

Postcondizioni: Il sistema torna allo stato principale.

UC5: Creare una configurazione compatibile con il motore semantico di Engagent

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione un comando per la creazione della configurazione.

Descrizione: L'utente, tramite CLI, crea una nuova configurazione.

Postcondizioni: É stato creato un nuovo file contenente la configurazione. Il sistema è tornato allo stato principale.

UC6: Caricare automaticamente la configurazione in Engagent

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione un comando per caricare automaticamente il file contenente la configurare in Engagent.

Descrizione: L'utente, tramite CLI, carica il file contenente la configurazione in Engagent.

Postcondizioni: Engagent contiene la nuova configurazione.

UC7: Inserire una configurazione già esistente

Attori Principali: Utente.

Precondizioni: Il sistema mette a disposizione un comando per l'input di una

configurazione già esistente (vengono ereditate *regole* e *synset*).

Descrizione: L'utente, tramite CLI, carica una configurazione esistente.

Postcondizioni: Il sistema contiene la configurazione di partenza.

3.2 Tracciamento dei requisiti

Da un'attenta analisi dei requisiti e degli use case effettuata sul progetto è stata stilata la tabella che traccia i requisiti in rapporto agli use case.
Il codice dei requisiti è così strutturato R[O/D][Num. requisito] dove:

R = requisito

O = obbligatorio

D = desiderabile

Tabella 3.1: Tracciamento dei requisti

Requisito	Descrizione	Fonte
RO-1	L'utente può inserire un file Excel	UC1
RO-2	L'utente può inserire un file Json	UC2
RO-3	L'utente può personalizzare la configurazione	UC3
RO-3.1	L'utente può modificare il dominio della configurazione	UC3.1
RO-3.2	L'utente può modificare la lingua della configurazione	UC3.2
RO-3.3	L'utente può modificare i prefissi delle regole nella configurazione	UC 3.3
RO-3.4	L'utente può modificare le priorità dei match nella configurazione	UC3.4
RO-4	L'utente può attivare o disattivare le funzionalità del programma	UC4
RO-4.1	L'utente può abilitare lo [g]stemming sulle categorie	UC4.1
RO-4.2	L'utente può disabilitare lo [g]stemming sulle categorie	UC4.1
RO-4.3	Valdiazione delle categorie, attraverso le frasi associate	UC4.2
RO-4.4	valdiazione delle categorie, attraverso le frasi associate	UC4.2
RO-4.5	Creazione dei *cluster*	UC4.3
RO-4.6	Creazione degli *intent*	UC4.5
RO-5	L'utente può creare un file contenente la configurazione	UC5
RO-6	le funzioni più importanti devono essere riutilizzabili	Tutor aziendale
RO-7	L'applicazione deve essere sviluppata in Python 3.7	Tutor aziendale
RD-1	creazione di una configurazione estendendone una già esistente	UC7
RD-2	caricare automaticamente una configurazione in *Engagent*	UC6
RD-3	esecuzione in tempo lineare rispetto alla grandezza del file	Tutor aziendale
RD-4	Il codice deve essere coperto da almeno 80% di test di unità	Tutor aziendale
RD-5	Il codice deve rispettare lo stile pep8	Obiettivo personale
RD-6	L'applicazione deve esporre un servizio API	Tutor Aziendale

Capitolo 4

Progettazione e codifica

4.1 Framework e tecnologie per lo sviluppo

Python

Python è un linguaggio di programmazione pensato per la ricerca. Per questo linguaggio sono stati sviluppati la maggior parte dei framework che trattano l'intelligenza artificiale (come TensorFlow). Questo vale anche per l'algoritmo di clustering presentato nell'introzione2.1.1.
Assieme al tutor aziendale, abbiamo scelto questo linguaggio per i seguenti motivi:

* permette di soddisfare il requisito di modularità del codice (requisitio RO-6): il codice prodotto è compatibile con quello dell'algoritmo di clustering;

* possiede dei framework per il pos-tagging e la generazione di sinonimi (NLTK e TreeTagger).;

nose

Test di unità e integrazione. Ambiente di test specifico di Python. L'estensione nose-cov permette di calcolare il code coverage.
Nella maggior parte dei casi, ho utilizzato il tool automatico di *Visual Studio Code* per eseguire i test. Con l'opzione di eseguire i test a ogni salvataggio, è possibile accorgersi subito se sono stati inseriti dei *bug*. Per calcolare il [g]code coverage è necessario eseguire il seguente comando:

```
$ nosetests --with-cov --cov src tests/
```

dove *src* è la cartella contenente il codice.

Figura 4.1: nose

pycodestyle (pep8) - pylint

Analisi statica del codice per Python. Rileva, a ogni salvataggio, errori di formattazione e di stile nel codice. Questi strumenti mi hanno permesso di mantenere un codice pulito, rispetto allo standard [g]PEP8.

Figura 4.2: Agile

NLTK

Framework di python per la creazione di sinonimi, attraverso dizionari italiani e inglese.

Figura 4.3: NLTK

TreeTagger

Applicazione per l'estrazione dei lemma dalle parole. Viene adattato in Python attraverso *TreeTaggerWrapper*.

Engagent

Piattaforma sviluppata da PAT s.r.l formata dalla chatbot, il motre semantico e servizi di supporto. É servita per eseguire i test di sistema e accettazione, in quanto target dei file di configurazione generati dalla applicazione sviluppata durante lo stage.

4.1.1 pip - PyPI

pip è un sistema di gestione di pacchetti di Python. Semplifica il processo di installazione, aggiornamento e rimozione di pacchetti Python, attraverso semplici comandi. Permette il tracciamento delle dipendenze utilizzate dal programma. Tramite *pip-env*, è possibile utilizzare un ambiente virtuale per garantire la portabilità dell'applicazione. *pip* mi ha permesso di eseguire eficientemente i seguenti task:

- individuazione, installazione e tracciamento dei pacchetti di Python;
- installazione dell'applicazione nel server aziendale remoto, contenente il sistema operativo Linux.

4.2 Progettazione

Durante lo stage, la progettazione è stata inserita all'interno del Manuale dello sviluppatore (documento in possesso di PAT s.r.l). Di seguito riporto tale progettazione, tralasciando però alcuni dettagli di implementazione, come richiesto dal tutor aziendale.

4.2.1 Struttura dell'applicazione

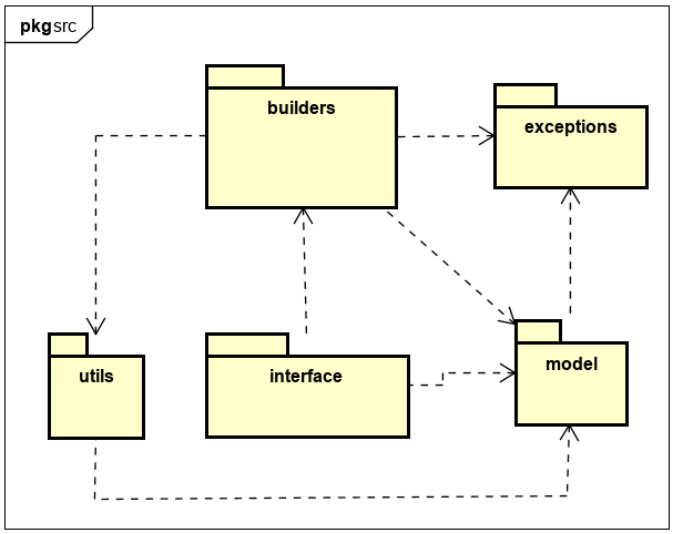


Figura 4.4: Diagramma dei package

model

Le classi in [g]model rappresentano una configurazione *NLP*.

NLP: Classe principale di *model*, rappresenta una configurazione NLP. Le altre classi sono dei componenti di questa. Il metodo principale è "to_string", che trasforma un oggetto NLP nella configurazione per *Engagent*.

Rule: Rappresenta una singola regola. Comprende il commento della regola, i *match*, le domande e le risposte.

Synset: Rappresenta un singolo synset. É composta da un titolo, alcuni valori di configurazione e i sinonimi.

Match: Rappresenta un singolo match di una regola. É composta da un insieme di categorie, la priorità del match e un titolo.

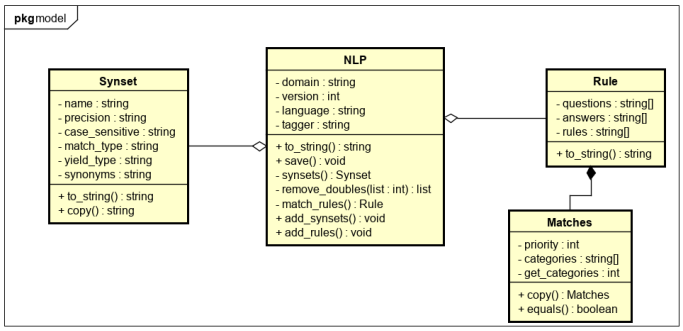


Figura 4.5: model

builders

Le classi in *builders* permettono di creare un oggetto NLP, senza preoccuparsi della logica di implementazione. Attraverso il design pattern *abstract method*, è possibile derivare la classe NLPBuilder, per creare builders che lavorano con formati diversi da JSON e XLSX.

NLPBuilder: Classe astratta per la creazione di un NLP. Contiene la logica principale di creazione delle configurazioni. Le classi che estendono questa classe devono solamente implementare i metodi astratti per standardizzare l'input (come definito nei commenti al codice e nel manuale dello sviluppatore).

NLPBuilderXLSX: Classe che estende NLPBuilder. Standardizza l'input nel formato xlsx (excel).

NLPBuilderJSON: Classe che estende NLPBuilder. Standardizza l'input nel formato json.

Nel diagramma è stato inserito anche il package *model* per specificare cosa crea ogni builder.

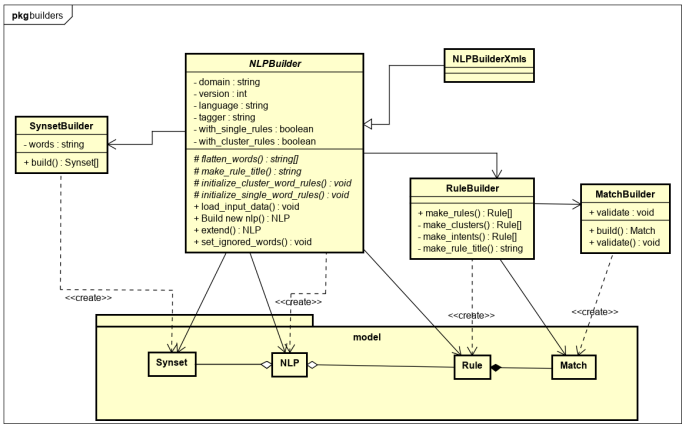


Figura 4.6: model

utils

Contiene moduli e classi di utilità.

SynsetGenerator: Questa classe contiene la logica di *business* del programma, ovvero quella che esegue la vera trasformazione dell'input in *synset* e regole (a differenza del model, che si limita a tradurre tali risultati in qualcosa di compatibile con Engagent). Questa classe utilizza la libreria NLTK e TreeTagger.

Utils: Modulo che contiene funzioni di utilità, utilizzate da più classi non dipendenti tra di loro.

NLPStemmer: Esegue lo stemming su un oggetto di tipo NLP.

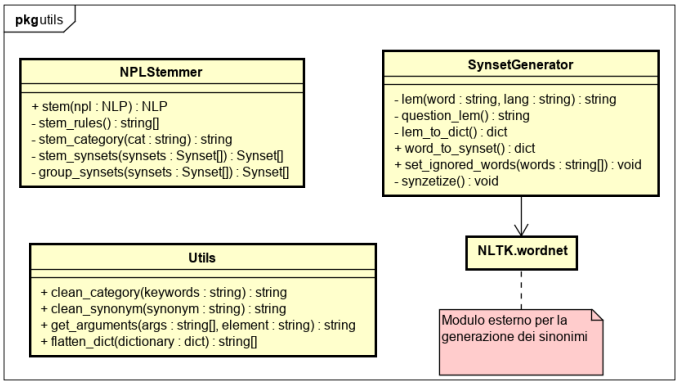


Figura 4.7: model

interface

Interfaccia dell'applicazione per l'utente. Permette l'interazione programmatica e a linea di comando.

api: Permette l'interazione programmatica con l'applicazione.

cli: Permette l'interazione a linea di comando con l'applicazione.

exceptions

Eccezioni personalizzate per l'applicazione.

EmptyCategoryException: Durante l'esecuzione del programma, è stata trovata una categoria vuota.

EmptySynonymException: Durante l'esecuzione del programma, è stato trovato un sinonimo vuoto.

4.3 Design Pattern utilizzati

L'applicazione è stata sviluppata utilizzando i seguenti design pattern:

* **Builder Pattern:** la creazione di un oggetto NLP può essere complicata, perché composta da almeno quattro componenti diverse. Il builder pattern permette di semplificare questo compito, rendendo di conseguenza le classi in *model* meno complesse;
* **Abstract Pattern:** permette di aggiungere nuovi formati in input all'applicazione senza dover riscrivere l'intera logica di creazione dell'NLP.

4.4 Codifica

4.4.1 Task della codifica

La codifica è stata intervallata con periodi di progettazione e analisi delle nuove richieste del tutor. Per ogni nuova componente da sviluppare, ho seguito questi passaggi:

* analisi e progettazione di dettaglio del problema;
* ricerca di soluzioni già esistenti per questo problema;
* codifica di quanto progettato e sviluppo di test di unità specifici;
* esecuzione di tutti i test di unità e risoluzione di eventuali *bug*;
* verifica da parte del tutor aziendale;
* risoluzione di eventuali errori logici.

4.4.2 Stile del codice

Per facilitare il lavoro di chi dovrà manutenere il progetto, ho seguito le linee guida definite in *PEP8*[1] per la stesura del codice:

* i metodi più significativi sono documentati con il seguente commento:

```
"""[Descrizione]

Arguments:
    arg1 {[tipo]} -- [descrizione]
Returns:
    [tipo] -- [descrizione]
Raises:
    [exception] -- [descrizione]
"""
```

* le variabili private iniziano con un doppio underscore '__';
* le variabili protette iniziano con un singolo underscore '_';
* le variabili sono scritte interamente in minuscolo, variabili composte da più parole sono seprarate da underscore (esempio: get_name)

[1] https://www.python.org/dev/peps/pep-0008/

4.4.3 Codice significativo

Di seguito, ho riportato degli esempi di codice significativo.

Implementazione di abstract method

L'*abstract method pattern* è stato utilizzato nella classe NLPBuilder. Il ruolo dei metodi astratti è lasciare alle classi derivate il compito di implementare la logica di trasformazione dell'input in un formato standard. Questo è il minimo neccessario richiesto per implementare un nuovo builder (per un nuovo formato di input).

L'implmenetazione del metodo *load_ input_ data* richiede la creazione di una variabile contenente il contenuto del file di input. Questa variabile verrà utilizzata solamente dalle implementazioni degli altri metodi astratti, quindi non è predefinita.

```
    @abstractmethod
    def load_input_data(self, path: str):
        """load the configuration from a file

        Arguments:
            path {str} -- relative path to the file
        """
        pass
```

Il metodo *_flatten_ words* estrae le categorie dall'input definito in *load_ input_ data*. Gli altri metodi astratti definiti in NLPBuilder seguono una logica simile.

```
    @abstractmethod
    def _flatten_words(self) -> list:
        """Flatten a list of categories in the input variable
           defined by load_input_data.

        Returns:
            words {list} -- of categories. E.g. ['dog','cat',
                plain','airplain']
        """
        pass
```

Capitolo 5

Conclusioni

5.1 Consuntivo

Questa sezione descrive i risultati dello stage sotto forma di metriche. Nel complesso, il progetto è stato concluso con successo.

5.1.1 Requisiti soddisfatti

L'applicazione soddisfa tutti i requisiti3.2 obbligatori (17/17) e la maggior parte dei requisiti desiderabili (5/6). Il requisito desiderabile non implementato, per questioni di tempo, è RD-6, ovvero esporre l'applicazione attraverso un'API. Implementerò questo requisito in un breve periodo di post-stage in azienda.

Tabella 5.1: requisiti soddisfatti

Requisito	**Risultato**
RO-1	soddisfatto
RO-2	soddisfatto
RO-3	soddisfatto
RO-3.1	soddisfatto
RO-3.2	soddisfatto
RO-3.3	soddisfatto
RO-3.4	soddisfatto
RO-4	soddisfatto
RO-4.1	soddisfatto
RO-4.2	soddisfatto
RO-4.3	soddisfatto
RO-4.4	soddisfatto
RO-4.5	soddisfatto
RO-4.6	soddisfatto
RO-5	soddisfatto
RO-6	soddisfatto
RO-7	soddisfatto
RD-1	soddisfatto
RD-2	soddisfatto
RD-3	soddisfatto
RD-4	soddisfatto

Tabella 5.1: requisiti soddisfatti

Requisito	Risultato
RD-5	soddisfatto
RD-6	non soddisfatto

5.1.2 Metriche del codice

Code coverage

Il *code coverage* è parti all'87%. I metodi non coperti sono molto semplici e non necessitano di essere testati.

```
---------- coverage: platform win32, python 3.7.2-final-0 ----------
Name                                                 Stmts   Miss  Cover
------------------------------------------------------------------------
src\__init__.py                                          0      0   100%
src\builders\__init__.py                                 0      0   100%
src\builders\match_builder.py                           24      3    88%
src\builders\nlp_builder.py                             97     10    90%
src\builders\nlp_builder_xlsx.py                        50      3    94%
src\builders\rule_builder.py                            66      8    88%
src\builders\synset_builder.py                          24      0   100%
src\exceptions\__init__.py                               0      0   100%
src\exceptions\exceptions.py                             6      0   100%
src\interface\__init__.py                                0      0   100%
src\interface\api.py                                    68     21    69%
src\interface\cli.py                                    35     35     0%
src\interface\config.py                                  5      0   100%
src\model\__init__.py                                    0      0   100%
src\model\match.py                                      17      1    94%
src\model\nlp.py                                        90      5    94%
src\model\rule.py                                       58      4    93%
src\model\synset.py                                     39      1    97%
src\wordnet_utils\__init__.py                            0      0   100%
src\wordnet_utils\config_to_nlp.py                      88      0   100%
src\wordnet_utils\nlp_stemmer.py                        93      9    90%
src\wordnet_utils\synset_generator.py                  103     12    88%
src\wordnet_utils\utils.py                              42      6    86%
------------------------------------------------------------------------
TOTAL                                                  905    118    87%
------------------------------------------------------------------------
Ran 12 tests in 13.111s
```

Figura 5.1: code coverage

Analisi statica

Durante la codifica, ho utilizzato *pycodestyle* per rispettare lo stile definito da PEP8. Questo ha portato a un codice privo di *warning* rilevati da questa applicazione.

```
PS C:\Users\stefano\Documents\pat-npl-generator> pycodestyle .\src\
PS C:\Users\stefano\Documents\pat-npl-generator>
```

Figura 5.2: pycodestyle

5.2 Raggiungimento degli obiettivi

5.3 Conoscenze acquisite

- **python**;
- ho potuto imparare nella pratica come funziona una *software house*, iniziando ad acquisire la professionalità richiesta per questo ambiente;

* ho imparato l'importanza e il ruolo del *pre-processing* dei dati, prima di essere elaborati da un algoritmo di machine learning. Nel mondo reale, i dati sono pochi e preziosi, quindi è necessario sfruttarli al massimo e assicurarsi che l'output dell'algoritmo di *machine learning* sia filtrato e analizzato, prima di essere esposto agli utenti del servizio.
  Questo si contrappone con l'esperienza universitaria, dove il focus principale viene posto sul metodo, invece che sul risultato.

5.4 Valutazione personale

Appendice A

Appendice A

Citazione

Autore della citazione

Bibliografia