**UNIVERSITÀ POLITECNICA DELLE MARCHE**
**FACOLTÀ DI INGEGNERIA**
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione


**TESI DI LAUREA**

# Progettazione e implementazione di un chatbot in tecnologia RASA per il supporto ai clienti di un gestore di servizi energetici

# Design and implementation of a chatbot in RASA technology for customer service of an energy services provider

Relatore

Prof. Domenico Ursino

Candidato

Leonardo Lucarelli

Correlatori

Ing. Massimo Callisto De Donato

Ing. Emiliano Anceschi

**ANNO ACCADEMICO 2020-2021**

*La scienza non è nient'altro che una perversione*
*se non ha come suo fine ultimo il miglioramento delle condizioni dell'umanità.*

Nikola Tesla

**Sommario**

Il presente lavoro intende fornire informazioni utili riguardo il mondo sempre più popolato dei chatbot. Partendo da nozioni teoriche relative al Natural Language Processing (NLP) e alle metodologie di sviluppo dei chatbot, il lavoro prosegue con la presentazione di un caso d'uso reale, descritto dall'analisi dei requisiti fino all'implementazione vera e propria tramite il framework RASA. Lo scopo dell'assistente è quello di fornire supporto ai clienti di un ente nel settore dei servizi energetici. In particolare, vengono determinati due ambiti distinti che costituiscono il dominio di competenza del chatbot: in primis, l'assistente deve essere capace di rispondere alle domande poste dall'utente riguardo i contenuti del sito web del committente. Il chatbot deve, inoltre, saper identificare la presenza di problematiche relative ai pagamenti degli utenti. Risulta evidente come l'architettura del sistema complessivo debba comprendere componenti che facciano uso di tecniche di Intelligenza Artificiale e Machine Learning e rendano l'intero lavoro altamente sperimentale.

**Keyword**: Chatbot; Assistente virtuale; RASA; Natural Language Processing; Intelligenza Artificiale; Machine Learning.

**Abstract**

This work's aim is to provide some useful information about the growing field of chatbots. Starting from the theory behind the Natural Language Processing (NLP) and the most important chatbot development techniques, this work continues with the description of a real case, from the analysis of requirements to the implementation through the RASA framework. The main purpose of such virtual assistant is to provide customer support to a company that works in the energy sector. In particular, two different goals are determined which define the domain in which the chatbot works. Fi,rst of all, the chatbot must be able to answer every user question regarding the content of the client's web site. Secondly, the assistant must be able to identify every possible problem related to users' payments. It's obvious that the overall system architecture includes components exploiting techniques of Artificial Intelligent and Machine Learning. For this reason, the work is considered to be highly experimental.

**Keywords**: Chatbot; Virtual Assistant; RASA; Natural Language Processing; Artificial Intelligence; Machine Learning.

# Indice

# Elenco delle figure

# Elenco delle tabelle

# Elenco dei listati

# Introduzione

Secondo molti, la storia dell'Intelligenza Artificiale inizia nel 1950 quando Alan Turing scrisse "Computing Machinery and Intelligence" per la rivista britannica Mind. All'interno di questo articolo Turing si pone il seguente quesito: "le macchine possono pensare?". La risposta, dopo diversi decenni, non è ancora chiara. L'Imitation Game, spiegato nell'articolo, prevede tre partecipanti: un uomo A, una donna B e una terza persona C, tenuta separata dagli altri due. Il gioco consiste nella determinazione, da parte del partecipante C, di chi è la donna e chi è l'uomo. Il Test di Turing prende spunto da tale gioco, sostituendo il soggetto A con una macchina. Tale test definisce, quindi, una macchina come intelligente se, essendo essa interrogata da una persona, quest'ultima non è in grado di determinare se la conversazione è avvenuta con un interlocutore umano o con una macchina.

Risulta evidente come i chatbot si prestino perfettamente a tale test; esso è, infatti, rapidamente diventato il test di riferimento per valutare la capacità di un chatbot di tenere conversazioni *human-like*.

Il primo chatbot, che prende il nome di ELIZA, è stato implementato nel 1966 da Joseph Weizenbaum. ELIZA ha lo scopo di simulare uno psicoterapeuta, restituendo spesso la frase immessa dall'utente in forma interrogativa. La sua abilità nella comunicazione è, quindi, limitata, ma è stata fonte di ispirazione per i successivi sviluppi di altri chatbot. Nonostante ciò, Weizenbaum fu stupito dalla reazione degli utenti. Benché sviluppò ELIZA come una caricatura di una conversazione umana, gli utenti iniziarono a confidare al chatbot i più profondi e bui segreti. Valutando ciò, gli esperti dichiararono che i chatbot sarebbero diventati indistinguibili dagli umani in pochissimi anni.

Il primo grande step in avanti è arrivato, in realtà, nel 1995 con ALICE (Artificial Linguistic Internet Computer Entity). Nonostante fosse basata sul pattern-matching come ELIZA, essa era capace di sostenere una conversazione in qualsiasi argomento. Tale differenza è conseguenza dello sviluppo di ALICE con il linguaggio Artificial Intelligence Marckup Language (AIML). Tale linguaggio ha permesso di estendere la knowledge base del chatbot dalle 200 keyword e regole di ELIZA a 41.000 template e pattern di ALICE. Tuttavia, ALICE non possiede feature "intelligenti" e non può esprimere emozioni o atteggiamenti tipici degli umani.

Per l'integrazione dell'Intelligenza Artificiale nello sviluppo dei chatbot, sarà necessario aspettare ulteriori 15 anni. Apple Siri (2010), IBM Watson (2011), Google Assistant (2012), Microsoft Cortana (2014) e Amazon Alexa (2014) sono attualmente gli assistenti vocali più popolari. Essi sono capaci di comprendere comandi vocali e compiere azioni che riguardano aspetti molto diversi tra loro (per esempio, accendere una lampadina o aggiungere un evento nel calendario). Nonostante il miglioramento rispetto ai chatbot sviluppati nel corso del

secolo scorso sia evidente, anche i chatbot di nuova generazione sono in grado di superare il Test di Turing solo in maniera ristretta, ossia ponendo domande solo nell'ambito in cui essi sono addestrati a rispondere.

Il presente lavoro si pone l'obiettivo di progettare e implementare un chatbot per fornire supporto clienti ad un ente che offre servizi in ambito energetico. Il Test di Turing passa, in questo contesto, in secondo piano. Nonostante in fase di progettazione ci si è posti l'obiettivo di sviluppare delle conversazioni naturali e umane, l'esigenza principale del progetto consiste nel garantire un'esperienza robusta e completa rispetto ai requisiti richiesti. In particolare, il chatbot dovrà essere in grado di rispondere alle domande dell'utente riguardo ciò che è contenuto nel sito web del cliente e, contemporaneamente, fornire assistenza agli utenti che riscontrano problematiche nei pagamenti dei servizi.

Già dai requisiti, si rende evidente come il sistema complessivo sia costituito da diversi componenti. In questa tesi vengono dettagliati ciascuno degli elementi facenti parte della soluzione, ponendo il focus sul chatbot. Infatti, quest'ultimo, fa da tramite tra l'utente e i modelli nel back-end garantendo, quindi, la comunicazione tra ognuno degli elementi del sistema.

Il chatbot è sviluppato in RASA, un framework open-source diventato velocemente lo standard per lo sviluppo di assistenti virtuali. La sede di svolgimento del presente progetto è stata Filippetti S.p.A la quale, su commissione, ha intrapreso un lavoro altamente sperimentale includendo tecniche di Intelligenza Artificiale (IA) e Machine Learning (ML) molto innovative.

L'intero progetto è stato coordinato utilizzando tecniche agili di progect management. Durante l'intera durata del lavoro si sono svolte molteplici riunioni con il cliente che, in maniera incrementale, ha espresso le proprie esigenze, fornito i dati richiesti per lo sviluppo e valutato i vari prototipi implementati indicando modifiche o miglioramenti. L'integrazione di un chatbot nel sito web del committente garantisce molteplici vantaggi. Il progetto nasce, infatti, come una Proof Of Concept (POC) con l'obiettivo di verificare le potenzialità delle moderne tecniche di IA e ML.

Il contenuto della tesi è strutturato di seguito specificato:

- Nel *Capitolo 1* verrà introdotto il Natural Language Processing (NLP) evidenziandone le caratteristiche e i metodi di implementazione. Inoltre, si mostrerà come tale tecnologia può essere applicata nei contesti aziendali.
- Nel *Capitolo 2* si descriverà lo stato dell'arte dei chatbot. Verranno anche dettagliate le caratteristiche e le funzionalità del framework che ha consentito lo sviluppo del chatbot, ovvero
- Nel *Capitolo 3* si entrerà nel cuore del progetto, introducendo il committente del lavoro e descrivendone gli obiettivi e i requisiti richiesti.
- Nel *Capitolo 4* verranno individuati le componenti del sistema necessarie per il soddisfacimento dei requisiti. Ogni elemento verrà dettagliato sia a livello tecnico che funzionale.
- Il *Capitolo 5* consisterà nella progettazione del ruolo del chatbot all'interno del sistema complessivo. In particolare, verranno individuati i flussi conversazionali e, quindi, le conversazioni che l'assistente sarà in grado di gestire.
- Nel *Capitolo 6* si mostrerà come le funzionalità previste in fase di progettazione sono state implementate nel framework RASA. Si dettaglierà, cioè, il contenuto di tutti i file che costituiscono il framework e come essi sono correlati tra loro.
- Nel *Capitolo 7*, infine, viene redatta la SWOT Analysis dell'intero lavoro per individuarne le qualità e gli elementi migliorabili.

# CAPITOLO 1

# Il Natural Language Processing

*Nel corso degli ultimi anni sono sempre più i dati che le medie e le grandi organizzazioni tendono a raccogliere. Il motivo di tale scelta è determinato dall'abbassamento dei prezzi dei dispositivi di storage e dal ruolo centrale che i dati hanno iniziato ad occupare all'interno di qualsiasi organizzazione. Infatti, se eleaborati, i dati forniscono informazioni che permettono l'ottimizzazione delle attività operative, l'estrazione di informazioni strategiche, l'identificazione di nuovi mercati, etc. Alla loro quantità crescente è corrisposta una crescita della potenza di calcolo delle macchine moderne; tuttavia,per affrontare la loro eterogeneità servono nuove tecniche di elaborazione e analisi. Una tra le importanti tecniche di elaborazione dei dati è il Natural Language Processing (NLP). Il presente capitolo si pone l'obiettivo di approfondire tale tecnica, le motivazioni per cui nasce e le possibili applicazioni.*

## 1.1 Cos'è il Natural Language Processing

Il Natural Language Processing è un ramo della Data Science il cui obiettivo è quello di permettere alle macchine di processare i testi e il linguaggio parlato. Tale operazione, banale per l'uomo, risulta per le macchine molto impegnativa, tanto che Alan Turing l'ha resa protagonista nel suo test di intelligenza per le macchine. Il motivo è dato dal fatto che queste ultime sono molto efficienti nell'elaborare dati strutturati, mentre l'oggetto di studio delle tecniche di NLP è un dato non strutturato.

### 1.1.1 Dati non strutturati

Non tutti i dati sono definiti in egual modo: suddividendoli in macrocategorie, è possibile classificarli in dati strutturati e dati non strutturati. I dati strutturati sono altamente organizzati e formattati in una struttura predefinita, vengono facilmente elaborati dagli algoritmi di Machine Learning e possono essere interrogati da linguaggi standardizzarti, come SQL. I dati non strutturati, al contrario, non possono essere processati e analizzati tramite metodi tradizionali. Infatti, essi non possiedono un modello predefinito: esempi di dati non strutturati sono testi, immagini, video o audio. Secondo una proiezione di Rydning [2021], entro il 2025 oltre l'80% dei dati globali sarà non strutturato e, ad oggi, solo lo 0.5% è analizzato. Si rende evidente come siano sempre più necessari metodi per lo studio e l'analisi automatica di questa tipologia di dato. Il presente capitolo è focalizzato sui dati non strutturati di tipo testuale.

### 1.1.2 Pipeline

La Figura 1.1 mostra un esempio di dato strutturato e di dato non strutturato, i quali rappresentano l'input e l'output di un generico algoritmo di NLP.

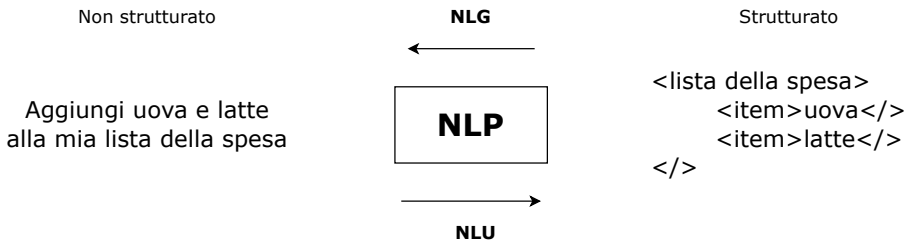


**Figura 1.1:** Esempio di un possibile funzionamento degli algoritmi di NLP

In particolare, si evidenzia la distinzione tra *Natural Language Generation* (NLG), cioè la traduzione da un dato strutturato in linguaggio naturale, e *Natural Language Understanding* (NLU), ossia tradurre in un dato strutturatouna o più frasi espresse in linguaggio naturale. Per gli scopi del presente lavoro, risulta più interessante approfondire come da un dato non strutturato sia possibile passare ad un dato strutturato, ossia analizzare le tecniche di NLU. Nella figura 1.2 vengono elencati i passaggi fondamentali di una pieline NLU.

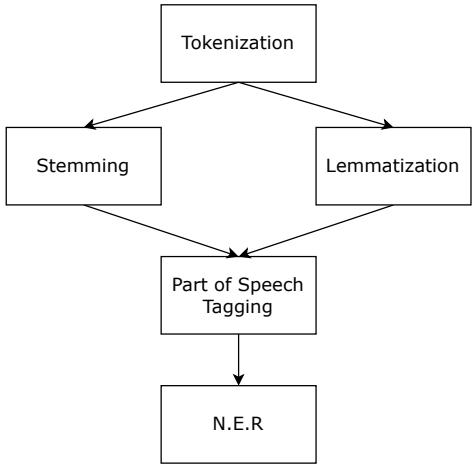


**Figura 1.2:** Alcune delle possibili componenti di una pipeline di NLU

Data una frase in input all'algoritmo, il primo step è la *tokenization*: si suddivide la stringa in token, cioè in parole, permettendo l'esecuzione dei successivi componenti su un token alla volta. Lo *stemming*, a seguire, deriva una parola da un certo token. Per esempio, le parole "aperto", "apre" o "aprirà" derivano tutte dalla stessa parola: "aprire". Lo *stemming*, però, non è efficiente su tutti i token: "universale" e "università" non derivano da "universo". La componente *lemmatization*, dato un certo token, ne estrae il significato attraverso la definizione dizionariale, derivandone la radice e risolvendo così le limitazioni dello *stemming*. La componente *part of speech tagging* esegue un controllo su dove il token in analisi è usato all'interno della frase, valutandone il contesto d'uso. Infine, il *natural entity recognition* restituisce, per ogni token, le eventuali entità associate. Per esempio, il token "Arizona" possiederà l'entità "Stato U.S." o il token "Jhon" avrà associata l'entità "Nome di persona". I componenti precedentemente elencati sono solo alcuni dei possibili strumenti applicabili per eseguire il Natural Language Understanding e permettono, quindi, la traduzione da un dato non strutturato a un dato comprensibile alle macchine.

### 1.1.3 Task

Il linguaggio umano è caratterizzato da ambiguità che rendono molto difficile progettare software in grado di determinare il vero significato di una frase scritta o parlata. Omonimi, sarcasmo, dialetto, metafore sono solo una parte delle irregolarità che l'uomo impega anni per imparare. Esistono molteplici task di NLP che frammentano il linguaggio umano in modo da aiutare le macchine a capire il senso di ciò che stanno analizzando. Alcuni esempi dei suddetti task sono:

- *Speech recognition*: anche chiamato speech-to-text, è il task che permette la conversione di dati vocali in dati testuali. Questo task è fondamentale per ogni tipo di applicazione che esegue comandi vocali, come gli assistenti virtuali.
- *Disambiguazione del significato di una parola*: per le parole con più di un significato, questo task seleziona quello più corretto tramite un processo di analisi semantica della frase, individuando il significato più attinente al contesto in cui la parola si trova.
- *Referenze multiple*: permette l'identificazione di se e quando due parole si riferiscono alla stessa entità. L'esempio più comune è la determinazione di persone od oggetti riferiti da un pronome, ma può anche includere l'identificazione di metafore presenti in un testo (e.g., "volpe" potrebbe intendere una persona molto furba, non l'animale).
- *Sentiment analysis*: Consiste nell'identificazione ed estrazione di opinioni dal testo.

La generalità di questi task ne permette l'applicazione in diversi ambiti; i più importanti sono affrontati nella prossima sezione.

## 1.2 Ambiti d'utilizzo

Gli ambiti commerciali che sfruttano le tecniche di NLP sono molteplici e diversificati: tutto ciò che richiede un'analisi di un dato testuale o vocale necessita una pre-elaborazione tramite il Natural Language Processing.

### 1.2.1 Sentiment Analysis

Come anticipato nella sezione 1.1.3, l'NLP può essere utilizzato per eseguire Sentiment Analysis. Tale strumento risulta molto utile nell'analizzare il linguaggio usato dagli utenti nei post e nei commenti del social network o nelle recensioni di prodotti. È possibile, quindi, estrarre attitudini ed emozioni legate ad un certo prodotto o argomento, così da permettere alle organizzazioni o aziende di effettuare modifiche, pianificare campagne pubblicitarie mirate e molto altro. Oltre all'ambito commerciale, è il contesto politico quello dove la sentiment analysis ha evidenziato le sue potenzialità. Infatti, analizzando l'atteggiamento dell'elettorato riguardo determinati argomenti, gli esponenti politici diventano capaci di guadagnare consenso aderendo alle esigenze dei cittadini. La sentiment analysis si è, inoltre, mostrata molto efficace nel predirre i risultati delle elezioni politiche: un esempio rilevante riguarda le presidenziali americane del 2016, dove gran parte degli studi sondaggistici tradizionali indicavano Clinton come vincitrice, mentre gli algoritmi di sentiment analysis applicati su decine di migliaia di contenuti dei social network previdero la vittoria di Trump.

### 1.2.2 Traduttori Linguistici

I traduttori linguistici sono un ulteriore esempio di applicazione delle tecnologie di NLP. Infatti, la vera traduzione da una lingua ad un altra non può avvenire semplicemente tramite una sostituzione di parole. Una traduzione efficace deve accuratamente estrarre il significato e il tono della lingua in input e tradurlo in un testo scorrevole con lo stesso significato e lo stesso impatto nella lingua in output. L'NLP permette anche traduzioni intralinguistiche: partendo da un dato testuale, è possibile eseguirne la parafrasi, il riassunto o l'estrazione di tag. Questi strumenti permettono l'indicizzazione di documenti, la condensazione di flussi continui di informazioni in blocchi oppure l'implementazione di un chatbot che esegue question answering efficacemente.

### 1.2.3 Chatbot

Con il termine *chatbot* si intende un software in grado di mantenere una conversazione coerente con uno o più utenti contemporaneamente. Anche i chatbot più sofisticati, per il loro funzionamento generale, integrano tecniche di NLP. L'obiettivo principale dei chatbot è, infatti, comprendere ciò che l'utente scrive e fornire una risposta consequenziale. I chatbot aiutano le aziende ad automatizzare molti processi, offrendo un servizio clienti e un'esperienza utente molto migliori. I chatbot possono essere implementati per due obiettivi principali:

1. *Marketing o promozionale*: il chatbot ha lo scopo di informare l'utente sui servizi o prodotti offerti dall'azienda, incrementare gli iscritti alla newsletter o guidare l'utente all'interno del sito. Tenendo traccia della navigazione dell'utente sul sito web, il chatbot sarà, inoltre, in grado di offrire suggerimenti di acquisto personalizzati.

2. *Servizio clienti*: è l'ambito dove i chatbot hanno il maggiore impatto. Le capacità di un chatbot spazionano dal fornire assistenza tecnica ai clienti alla prenotazione di un servizio (ad esempio, riservare un tavolo in un ristorante o una stanza in albergo), ma anche fornire informazioni riguardo ordini e spedizioni o semplicemente collezionare le informazioni di un utente per facilitare il lavoro un agente di supporto umano.

I chatbot garantiscono il loro servizio in qualsiasi orario e in qualsiasi giorno, assicurando una risposta immediata. Ciò procura alle aziende molteplici vantaggi: incremento delle vendite, riduzione dei costi di acquisizione di nuovi clienti e aumento della visibilità del brand. Avendo ora chiari i casi d'uso di questa tecnologia, nel capitolo successivo si mostreranno i metodi d'implementazione.

CAPITOLO 2

# I chatbot e la tecnologia RASA

*Questo capitolo vuole offrire una panoramica sui chatbot, le possibili architetture per la loro implementazione e i conseguenti vantaggi e svantaggi di ogni approccio. Inoltre, verrà descritto il framework RASA in tutte le sue caratteristiche e funzioni. Tale framework, infatti, è rapidamente diventato lo standard open source per l'implementazione dei chatbot ed è la piattaforma scelta per lo sviluppo del presente progetto.*

## 2.1 Tipologie

I vantaggi commerciali dell'avere un chatbot, descritti nel capitolo precedente, spingono molte aziende a investire in questa tecnologia. Nel momento in cui un'azienda decida di usufruirne, in base all'utilizzo che prevede di farne e agli obiettivi che vuole raggiungere, ha a disposizione due differenti approcci di implementazione: i rule-based chatbot e gli AI-based chatbot. La suddivisione nelle suddette macrocategorie deriva dalle omonime tecniche di NLP impiegate dai chatbot per comprendere o generare testo in linguaggio naturale.

### 2.1.1 Rule-based chatbot

Il loro funzionamento è guidato da un insieme di regole, determinate in fase di sviluppo, che definiscono quali problemi il chatbot è in grado di gestire fornendo le opportune soluzioni. Al di fuori di tali regole, però, il chatbot non può fornire risposte a nessuna domanda, avendo buone prestazioni solo nel dominio in cui è stato allenato. Un approccio molto semplice per la definizione di regole, mostrato in Figura 2.1, è dato dalla concatenazione di istruzioni IF-THEN-ELSE con risposte predefinite.

I chatbot rule-based, quindi, tracciano le conversazioni in anticipo: il progettista deve determinare cosa l'utente potrà chiedere e come il chatbot dovrà rispondere, generando, così, un flusso conversazionale ad albero. Questa tipologia di chatbot è molto efficiente nel rispondere a semplici domande, come prenotare un tavolo in un ristorante o richiedere gli orari di apertura di un negozio, ma spesso i chatbot appartenenti a tali categorie forniscono un'interazione molto meccanica e poco umana. I chatbot rule-based, quindi, permettono un flusso conversazionale poco flessibile, offrendo all'utente un'esperienza definita a priori dal progettista. Al contrario, i chatbot che fanno affidamento sul Machine Learning, risultano essere molto meno prevedibili.

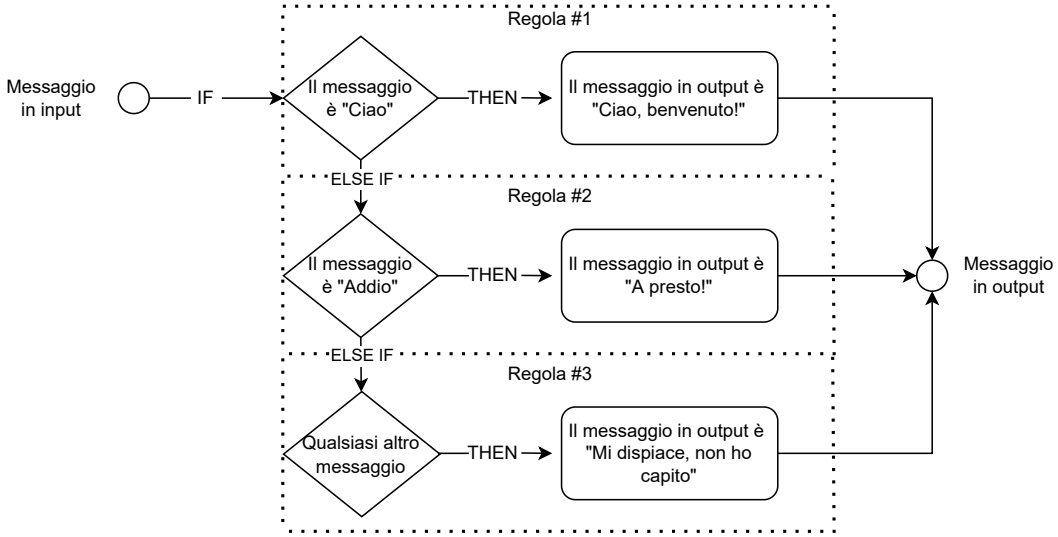


**Figura 2.1:** Semplici regole per un rule-based chatbot

### 2.1.2 AI-based chatbot

A differenza dei rule-based, i quali si servono di key word per il task di Natural Language Understanding, i chatbot basati sull'Intelligenza Artificiale comprendono il contesto e l'intento di una domanda prima di formulare una risposta. Un'architettura di questo tipo è solitamente in grado di gestire un grande numero di problematiche, anche molto complesse. Inoltre, gli AI-based chatbot possono rispondere a domande con errori di battitura o grammaticali e permettono uno sviluppo e un miglioramento incrementale basandosi sulle conversazioni già sostenute. Nonostante questa tipologia di chatbot richieda un ampio dataset per il training, essa permette di sviluppare un assistente con un suo specifico carattere: è possibile, infatti, decidere il livello di empatia, umorismo o cordialità. Inoltre, proprio perché alimentati dall'AI, i chatbot appartenenti a questa categoria possono muoversi da un contesto all'altro: esempio cardine sono i generative chatbot, di tipo open domain, che generano una risposta da zero in qualsiasi argomento che l'utente introduce.

## 2.2 RASA

RASA è un framework open source per lo sviluppo di assistenti contestuali, cioè capaci di capire il contesto delle conversazioni; in questa tipologia di chatbot cosa l'utente ha precedentemente scritto e come lo ha fatto influenza il modo in cui la conversazione prosegue. Considerare il contesto significa anche essere in grado di comprendere e rispondere ad input inaspettati, guidando l'utente verso il flusso conversazionale previsto quando egli se ne allontana.
Inoltre, RASA è progettato per l'implementazione di un dialogue system di tipo task oriented: l'utente ha un compito da eseguire e, parlando con un sistema automatico in una conversazione bidirezionle, il dialogue system ne permette il raggiungimento.

### 2.2.1 Descrizione del framework

Il framework, basato su Python, permette la creazione di chatbot tramite tecniche di machine learning supervisionato. Essendo un settore molto richiesto, le alternative a RASA

sono molteplici: aziende come Google, Microsoft e Amazon offrono piattaforme proprietarie per lo sviluppo di chatbot. La scelta è ricaduta su RASA per molteplici motivi:

- *Open source*: come anticipato, il cuore di RASA è totalmente open source. Ciò permette l'accesso e la modifica del codice del framework stesso: tale caratteristica è stata valorizzata nel presente lavoro, come descritto nella Sezione 6.2.1. La sua natura open source permette a RASA di essere trasparente. Molti provider offrono soluzioni "black box" che nascondono il funzionamento interno. Usare tali soluzioni limita il controllo sul comportamento del sistema da parte degli sviluppatori impedendo modifiche al framework o l'aggiunta di nuove capacità e use cases.

- *Sviluppo continuo*: come conseguenza all'architettura open source, si è formata una grande community i cui feedback vengono raccolti dagli sviluppatori di RASA. Tale fattore ha come effetto un continuo miglioramento del framework, con tempi di risoluzione dei bug e sviluppo di funzionalità utili molto contenuti.

- *Local Mode*: RASA permette l'esecuzione in locale, senza passare attraverso servizi cloud. Ciò si dimostra estremamente importante per garantire la sicurezza. Gli utenti, infatti, potrebbero fornire dati sensibili al chatbot: mantenere questi dati nello stesso server dove il chatbot è in esecuzione assicura una maggiore sicurezza e privacy.

Tali caratteristiche rendono RASA il framework open source per lo sviluppo di assistenti virtuali più popolare.

### 2.2.2 Architettura

L'architettura di RASA è un modello ibrido tra un approccio di tipo rule-based e uno orientato all'Intelligenza Artificiale. Per quanto riguarda il task di Natural Language Understanding (NLU), la componente rule-based è soprattutto utilizzata per riconoscere espressioni regolari nei messaggi, come indirizzi email o numeri di telefono. Essendo questi dati con una struttura ben definita, la loro estrazione è molto veloce e non necessita di un training set per operare. La limitazione di questo approccio è che, tramite tali tecniche, non è possibile gestire pattern non visti precedentemente. Tale limite è superato dall'integrazione di un approccio di tipo neurale che richiede un ampio training set ma che rende l'intera struttura molto più flessibile.
Allo stesso modo, una volta compreso ciò che l'utente ha scritto, anche la decisione del prossimo step è di tipo ibrido. L'approccio rule-based può essere paragonato ad una struttura ad albero che comprende tutte le possibili conversazioni, ma è tipico di chatbot che fanno uso di pulsanti nei quali le capacità dell'assistente sono ben delineate e si vuole mantenere l'utente all'interno di tali confini. Come atteso, però, questi chatbot hanno problemi nelle generalizzazioni e sono difficili da mantenere ed espandere. L'approccio neurale sceglie il prossimo step basandosi sull'intera conversazione sostenuta fino a quel momento e sulle conversazioni viste durante la fase di allenamento.

I componenti del framework sono descritti nella Figura 2.2 e dettagliati nel seguito.
Ne vengono ora descritte le funzioni:

- *Tracker Store*: memorizza le conversazioni nel disco tramite file, tali conversazioni possono essere salvate anche in un database SQL, MongoDB, o tramite altre opzioni.

- *Lock Store*: garantisce che i messaggi siano processati nel giusto ordine, permettendo a Rasa di essere eseguito in parallelo in diversi server. Gli utenti possono interfacciarsi con nodi diversi durante la conversazione e il suo proseguirsi non ne è influenzato.

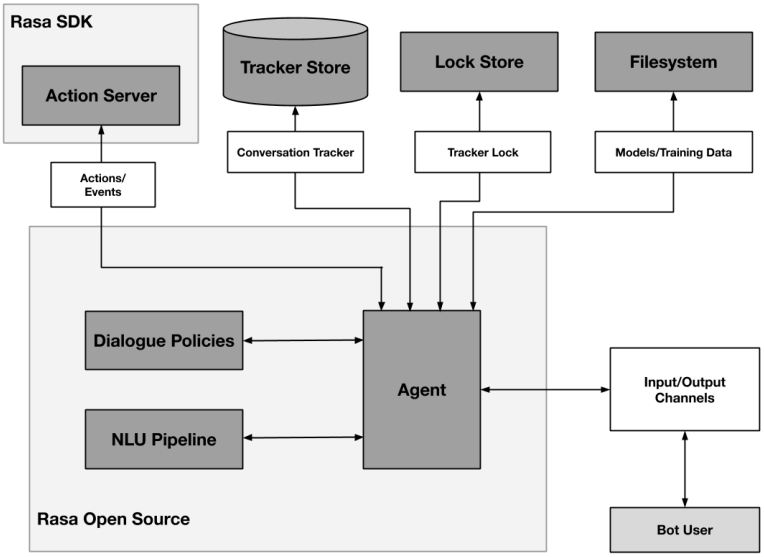


**Figura 2.2:** Struttura architetturale del framework

- *Filesystem*: il modello, una volta allenato, può essere salvato insieme ai dati di training in diversi posti, ovvero nel disco, in un server, in cloud, etc.
- *Action Server*: esegue le custom action, ossia quelle azioni che richiedono codice Python. Quando l'assistente prevede una custom action, il server invia una richiesta POST all'action server. Quando quest'ultimo termina l'esecuzione, restituisce un payload JSON di risposte ed eventi. Rasa SDK è il Software Development Kit in Python per eseguire le custom action.

Infine, gli elementi principali che costituiscono RASA Open Source sono i seguenti:

- *NLU Pipeline*: è la componente che permette la comprensione di ciò che l'utente scrive. Per garantire alte performance anche in assenza di dati di training, la pipeline può integrare dizionari open source, ad esempio Spacy. Proprio come i classici algoritmi di NLU, il compito della pipeline è ricevere in input un dato non strutturato (ossia il messaggio scritto dall'utente) ed estrarre dati strutturati sotto forma di intenti e entità. Gli intenti possono essere visti come etichette sui messaggi dell'utente che indicano l'obiettivo del messaggio stesso. Le entità sono pezzi di informazione che possono essere utili all'assistente.
  Per esempio, al messaggio: "Voglio prenotare un tavolo per un ristorante cinese", corrisponderà l'intento "`prenotazione`" e l'entità "`ristorante cinese`".
- *Dialogue Policies*: anche chiamato RASA Core, questo componente viene attivato quando le attività di NLU sono terminate. In base agli intenti e alle entità estratte e conoscendo lo storico della conversazione sostenuta fino a quel momento, le Dialogue Policy hanno il compito di decidere quali azioni il chatbot debba intraprendere. Ad ogni scambio, le policy definite nel file di configurazione predicono la prossima azione con un certo livello di confidenza in base ai pattern appresi in fase di allenamento.

Lo schema in igura 2.3 descrive come RASA elabora il messaggio ricevuto dall'utente.

Il messaggio è ricevuto e passato all'`Interpreter`, che lo converte in un dizionario che include il testo originale, l'intento e le eventuali entità estratte. Tale compito è gestito dalla pipeline di NLU. Il messaggio così generato è passato al `Tracker`, che tiene traccia

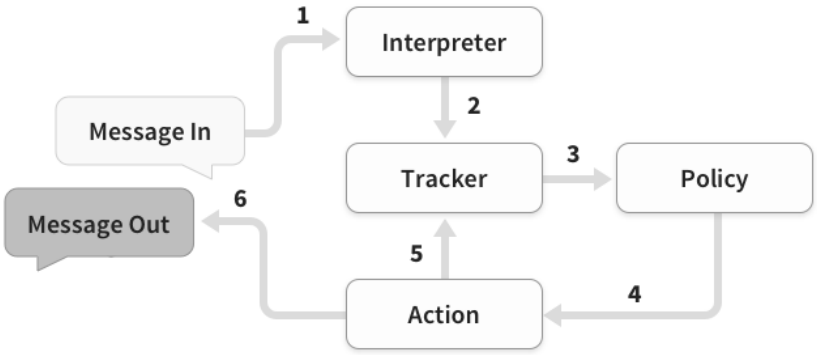


**Figura 2.3:** Fusso di elaborazione dei messaggi in RASA

della conversazione. Lo stato corrente del tracker è inviato a tutte le policy definite nel file di configurazione e viene predetta la prossima azione che l'assistente deve compiere. L'azione viene registrata nel `Tracker` e la risposta viene inviata all'utente.

### 2.2.3 File di configurazione

RASA predispone una moltitudine di file di configurazione, ciascuno dei quali ha la sua specifica funzione.

**Domain**
È il principale file del framework che definisce l'universo dove l'assistente è in grado di operare. Esso contiene i nomi degli intenti e delle entità che la pipeline di NLU dovrà estrarre, gli slots, i nomi delle custom actions e le risposte. Si descrive nel dettaglio questo convetto:

- *Intent*: rappresentano il significato del messaggio inviato dall'utente, cioè lo scopo che egli vuole raggiungere. Per esempio, il messaggio "Ciao" avrà come intent "Saluto". All'interno del file `Domain.yml` vengono definiti soltanto i nomi degli intenti. Ogni intento avrà associati più esempi definiti in `/data/nlu.yml`, descritto nel seguito. Quando il chatbot è in esecuzione, viene eseguito un task di classificazione; il messaggio ricevuto viene elaborato dalla pipeline di NLU e viene estratto un intento con la sua relativa confidenza. Tramite una configurazione personalizzata, è possibile anche estrarre due o più intenti per messaggio.

- *Entity*: sono porzioni di messaggio che, se estratte, aiutano l'assistente nel proseguimento della conversazione. Esempi di entità possono essere il nome dell'utente, l'indirizzo email o altre informazioni personali. Tale tipo di entità permette al chatbot di sostenere una conversazione più umana; se l'utente dichiara in un messaggio il proprio nome, salvarlo e riutilizzarlo in un interazione successiva ha un impatto molto positivo sulla sua user experience. Altro tipologia di entità sono quelle funzionali: ad esempio, al messaggio "Cerco un albergo a Venezia", corrispondono le entità "albergo" e "Venezia". L'estrazione di tali entità è indispensabile per portare a termine il task per cui il chatbot è stato progettato. A differenza degli intenti, ogni messaggio può contenere zero, una o molteplici entità.

- *Slot*: consistono nella vera e propria memoria del framework. Il loro utilizzo principale è quello di memorizzare i valori di alcune delle entità estratte, per poi poterli riutilizzare in qualsiasi punto della conversazione.
- *Form*: molto utili quando il task che l'assistente deve compiere richiede un certo numero di informazioni predeterminate, le form rappresentano l'insieme degli slot che il bot deve valorizzare per compiere il task. Nel momento in cui una form si attiva, il framework entra in loop; finché l'utente non fornisce tutte le informazioni richieste (o nega di volerle fornire), l'assistente pone domande per "valorizzare" gli slot vuoti che compongono la form. Ritornando all'esempio dell'albergo, per poter effettuare la prenotazione, il chatbot potrebbe aver bisogno di conoscere il numero delle persone, la città in cui si vuole alloggiare e il periodo di permanenza. Se l'utente scrive "Cerco un albergo a Venezia", lo slot relativo alla città verrà valorizzato automaticamente e il bot chiederà soltanto le informazioni riguardanti il numero dei partecipanti e le date di permanenza.
- *Response*: una volta estratto l'intento, le Dialogue Policies possono prevedere come prossima azione una risposta che non richiede codice Python per la sua esecuzione. Tali risposte sono definite nel `Domain` e, per convenzione, il nome è di tipo `utter_nome_risposta`. Una risposta può essere solo testuale o contenere immagini, può integrare pulsanti o includere più opzioni. Ad un'unica risposta, per esempio `utter_saluto`, possono corrispondere repliche diverse, per esempio "Ciao!" e "Buongiorno". A run-time RASA opzionerà una delle due risposte in modo casuale, ottenendo conversazioni leggermente diverse anche se con lo stesso pattern.
- *Actions*: Qui risiede l'elenco dei nomi delle custom action, il cui codice è contenuto nel file `/actions/actions.py`

**NLU**
Questo file contiene i dati di training della pipeline di NLU. Il modello, infatti, richiede un apprendimento di tipo supervisionato. Il file è costituito da un elenco di intenti, ognuno dei quali composto da un certo numero di esempi. Il Listato 2.1 rappresenta un'istanziazione di un intento con 3 esempi. La sintassi prevede l'inserimento di parole tra parentesi quadre per indicare il valore dell'entità e tra parentesi tonde per riferirsi al nome dell'entità da valorizzare.

```
nlu:
- intent: prenotazione
  examples: |
    - vorrei prenotare un albergo a [Venezia](luogo)
    - voglio andare a [Firenze](luogo) per [tre giorni](durata)
    - mi serve un hotel
```

**Listato 2.1:** Esempio di definizione di un intento

Un particolare tipo di intento sono quelli `retrieval`. Gli intenti così definiti, che godono di una loro sintassi distinguente, possono essere suddivisi in sotto-intenti. In questo lavoro gli intenti retrieval sono stati utilizzati per separare l'intento `chitchat` (cioè chiacchiere al di fuori del contesto del chatbot) in intenti più piccoli, permettendo così una risposta ad hoc per ogni diversa chitchat individuata.

**Rules**
Se tra le `Dialogue Policy` viene inclusa la `RulePolicy`, che ha priorità maggiore rispetto alle altre policies, allora è possibile definire delle regole all'interno di questo file. Le regole,

come intuibile, sfruttano la componente rule-based del Core di RASA. In diverse situazioni, infatti, ad uno specifico intento corrisponde un'azione o risposta determinata. Un esempio di regola è mostrato nel Listato 2.2.

```
rules:
- rule: nlu_fallback
  steps:
    - intent: nlu_fallback
    - action: utter_riformula
```

**Listato 2.2:** Esempio di definizione di una regola

L'`nlu_fallback` è un intento predefinito di RASA che viene attivato quando la confidenza sulla predizione di tutti gli intenti definiti nel file `Domain` non supera una soglia specifica. In altre parole, quando la pipeline di NLU non è in grado di classificare l'intento del messaggio scritto dall'utente, invia alle Dialogue Policies l'intento fallback. Tramite la `rule` sopra definita, ogni qual volta la `Rule_Policy` riceve l'intento di fallback, invia in output il contenuto della risposta `utter_riformula`, definita nel `Domain`.

**Stories**
Insieme al file di Rules, è costituito dai dati di training per il Core di RASA. A differenza delle regole, però, questo tipo di dati rappresenta una vera e propria conversazione sotto forma di alternanza di intenti e risposte o azioni. In questo file, quindi, si vanno a progettare le possibili conversazioni che l'utente può sostenere con l'assistente. Vanno previsti sia gli `happy path`, cioè quando l'utente fornisce le informazioni richieste e segue il pattern previsto, che gli `unhappy path`, ossia quando l'utente devia dal percorso atteso.

**Config**
Qui è dove è definita la Pipeline di NLU e le Dialogue Policies. RASA offre diverse configurazioni predefinite, ma garantisce piena libertà di modifica.
Gli elementi principali che costituiscono la Pipeline di NLU sono:

- *Modelli di linguaggio*: possono essere visti come dizionari. Sono modelli pre-allenati che è possibile integrare nella pipeline e hanno il compito di generare word vectors. Ne esistono molteplici e disponibili in diverse lingue e permettono di avere un numero di dati di training più contenuto.
- *Tokenizer*: suddividono un messaggio in token. In base alla lingua i token sono creati in modo differente: per l'italiano (e le lingue occidentali in generale) i token sono generati dal carattere "spazio". Tale operazione risulta necessaria per permettere l'esecuzione dei componenti successivi della pipeline.
- *Featurizer*: prende in input un dato grezzo (il token) e lo trasforma in un insieme di feature. Con feature si intende una rappresentazione del dato comprensibile dalle macchine usata come input per modelli di machine learning.
- *Intent Classifier e Entity Extractor*: sono dei veri e propri modelli. Il primo esegue un'operazione di classificazione multiclasse sull'input dell'utente: ogni messaggio verrà etichettato con uno (o più) degli intenti definiti nel Domain, con associata una confidenza sulla classificazione. L'estrazione delle entità, invece, avviene tramite dei modelli che possono anche essere preallenati. Esistono, infatti, degli estrattori specifici per entità come nomi di persone, numeri o date. È comunque possibile usare estrattori generici che verranno allenati con i dati di training contenuti in `data/nlu.yml`.

Per quanto riguarda le Dialogue Policies, queste vengono descritte di seguito, ordinate per priorità crescente:

- *TEDPolicy*: la Transformer Embedding Dialogue (TED) Policy è un'architettura multi task per la predizione della prossima azione. Consiste di diversi encoder che prendono in input le feature relative agli intenti, entità, slot e le azioni eseguite in precedenza e generano un unico feature vector. La prossima azione è, quindi, decisa cercando di massimizzare la similarità tra il feature vector generato e i vettori che rappresentano le possibili prossime azioni.
- *UnexpecTED Intent Policy*: permette all'assistente di reagire a input non previsti. Ha la stessa struttura della `TEDPolicy` ma, al posto di predire la migliore azione da compiere, impara quali sono i set più probabili di intenti che l'utente potrebbe esprimere basandosi sui dati contenuti nel file `data/stories.yml`. È usata solo in combinazione con altre policy.
- *Memorization Policy*: controlla se la conversazione corrente coincide con una delle storie definite nei dati di training. Se così fosse, predice la prossima azione con una confidenza di 1.0, altrimenti esegue una predizione nulla.
- *Rule-based Policies*: gestisce le parti di conversazioni che seguono un comportamento fisso. Le predizioni vengono effettuate basandosi sulle regole definite nel file `/data/rules.yml`.

# CAPITOLO 3

# Contesto di riferimento e descrizione dei requisiti

*Il presente capitolo si pone l'obiettivo di descrivere gli scopo del progetto. Per far ciò, si descrive brevemente l'azienda che ha commissionato il lavoro e i risultati che si vogliono ottenere dallo sviluppo del chatbot. Successivamente, si andranno a dettagliare i requisiti individuati in fase di progettazione.*

## 3.1 Il cliente

Al gruppo Filippetti, sede di svolgimento del tirocinio, è stata richiesta una Proof Of Concept (POC) di un chatbot da parte di un fornitore di servizi energetici.

### 3.1.1 Identità dell'azienda

Il cliente si definisce come un promotore dello sviluppo sostenibile in Italia. Tramite quest'azienda, infatti, l'Italia investe nella promozione della sostenibilità ambientale e nella costruzione di un'economia a basso contenuto di carbonio. Il cliente utilizza strumenti di incentivazione per cittadini, imprese e Pubbliche Amministrazioni per permettere il raggiungimento degli obiettivi comunitari e nazionali in materia di fonti rinnovabili ed efficienza energetica.

Gli obiettivi sfidanti per il 2030 individuati dall'Unione Europea consistono in una riduzione della CO2 del 40%, un miglioramento dell'efficienza energetica del 30% e un aumento dei consumi da fonti rinnovabili fino al 27%. Gestendo una moltitudine di servizi, il cliente sostiene la produzione elettrica della quasi totalità degli impianti a fonti rinnovabili in Italia, incentiva interventi per l'incremento dell'efficienza energetica e per la produzione di energia termica e si impegna nell'elaborazione di studi, ricerche e in attività di supporto alla Pubblica Amministrazione.

L'obiettivo della POC è quello di verificare se un chatbot all'interno del sito facilita la promozione della culutura della sostenibilità ambientale e dell'efficienza energetica. Inoltre, si vuole alleggerire il lavoro degli agenti di supporto, mettendo a disposizione strumenti per l'individuazione di problematiche nelle pratiche attive dei clienti.

## 3.2 Analisi dei requisiti

Come anticipato, i requisiti richiesti possono essere suddivisi in due macrocategorie distinte che, nel seguito verranno denominati ambiti. Essi sono:

- *Ambito Promozionale*: lo scopo del chatbot in questo ambito è quello di promuovere una consapevolezza nelle scelte, sia per la produzione di energia elettrica e/o termica, sia per l'efficienza energetica.
- *Ambito Customer Care*: l'obiettivo dell'assistente è quello di creare nuovi servizi a valore aggiunto per il cittadino, ottimizzare i processi aziendali in termini di efficacia ed efficienza interna (ad esempio, riduzione dei costi operativi) ed esterna (ad esempio, riduzione dei tempi di risposta).

### 3.2.1 Descrizione dei requisiti

Il numero delle visite e interazioni nel sito del cliente negli ultimi anni ha subito un forte aumento; per mantenere questo trend l'azienda ha deciso di impiegare l'Intelligenza Artificale e i Big Data in un progetto di miglioramento dell'esperienza del cittadino con la propria organizzazione e nell'efficientamento dei processi. I requisiti descritti nel seguito sono frutto di diversi incontri con i responsabili tecnici del cliente, i quali hanno definito incrementalmente le specifiche che il chatbot deve possedere e i domini nei quali l'assistente deve muoversi. Nel seguito si dettagliano i due ambiti.

**Ambito Promozionale**
La sperimentazione richiesta consente all'azienda cliente di approfondire le potenzialità delle tecnologie di AI e Big Data per impiegarli consapevolmente nel servizio di promozione al fine di migliorare il rapporto con i suoi fruitori dei servizi. L'utilizzo di un assistente virtuale consentirà l'adozione di un linguaggio naturale per fornire risposte corrette e non obsolete agli interlocutori (PA, imprese e cittadini) del cliente. In particolare, per ogni categoria di utente, il chatbot avrà uno scopo definito:

- *Per la Pubblica Amministrazione* lo scopo è quello di accelerare la transizione energetica, la riqualificazione energetica delle opere pubbliche, etc.
- *Per le imprese* l'obiettivo è quello di aiutarle nel consolidamento di una nuova cultura dell'energia e nell'implementazione di modelli di sviluppo, che integrino la dimensione economica, ambientale e sociale e generino nuove forme di business piùù sostenibili.
- *Per i cittadini* lo scopo è quello di fornire informazioni tempestive e aggiornate sugli strumenti di sostegno previsti per loro.

Il dominio di competenza del chatbot è, quindi, molto esteso. In particolare, si richiede che esso sia in grado di rispondere a domande generiche (o specifiche) su parte dei contenuti del sito web, sia contenuti testuali i link a documenti pdf. Sono stati individuati due macro-componenti di AI che dovranno interagire tra di loro: il chatbot e un modello di neural search. Il flusso dei processi base che ci aspettiamo è rappresentato nella Figura 3.1. Il chatbot, oltre a gestire la conversazione iniziale, avrà l'ulteriore compito di guidare l'utente attraverso una serie di scelte iniziali che avranno lo scopo di focalizzare ulteriormente il dominio specifico dell'argomento che egli vuole approfondire.

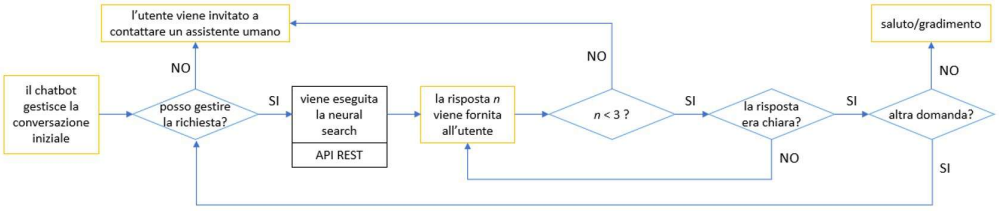


**Figura 3.1:** Flusso conversazionale atteso

L'idea è quella di fare una serie di domande all'utente in modo da individuarne la tipologia e la macro area della ricerca. Le informazioni raccolte dal chatbot verrano, quindi, usate successivamente in fase di neural search per filtrare opportunamente la base di dati utilizzata nella formulazione della risposta ai quesiti.

**Ambito Customer Care** In questo ambito si vuole alleggerire il carico di lavoro degli agenti umani del Contact Center. I flussi di interazione, dettagliati per tipologia di canale, sono mostrati nella Tabella 3.1.

| | Numero nel 2020 | Media Mensile 2020 | Media Giornaliera 2020 |
|---|---|---|---|
| Telefono | 180.274 | 15.023 | 707 |
| E-mail | 12.805 | 1.067 | 50 |
| Portale di Supporto | 121.184 | 10.099 | 475 |
| Area Clienti | 28.627 | 2.386 | 112 |
| Prioritario | 1.244 | 104 | 5 |
| Canali derivati | 23.950 | 1.996 | 94 |
| Outbound telefonico mail | 2.453 | 204 | 10 |
| Callback telefonico da IVR | 48.219 | 4.018 | 189 |
| Totale volumi | 429.176 | 35.756 | 1.683 |

**Tabella 3.1:** Andamento del numero di contatti per canale

I dati evidenziano una preferenza del canale "telefono" che manifesta l'esigenza di soluzioni chatbot sempre più user-oriented. A tal proposito, nella Figura 3.2 sono mostrati i requisiti delle interfacce evolute. Usabilità, comprensibilità, originalità, gradevolezza ed emozione sono attributi in larga misura indipendenti. Un sistema può avere un'interfaccia comprensibile e non essere usabile, può essere gradevole ma non suscitare emozioni, oppure può godere contemporaneamente di tutte queste caratteristiche.

Affinché un'interfaccia abbia una buona usabilità è necessario considerare le seguenti caratteristiche:

- *Chiarezza*: l'attenzione dell'utente è diretta verso l'informazione necessaria che deve essere chiaramente comprensibile, facile da leggere e non ambigua.
- *Concisione*: agli utenti viene fornita solo l'informazione necessaria per eseguire il compito.
- *Consistenza*: la medesima informazione è presente all'interno del sistema, conformemente alle aspettative dell'utente.

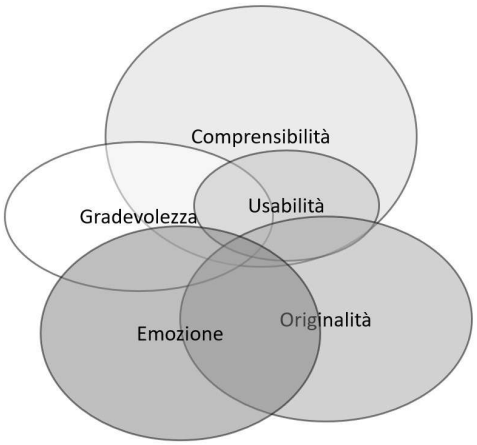


**Figura 3.2:** Requisiti della grafica di un sistema interattivo

In ambito customer care, l'assistente deve essere in grado di eseguire i seguenti task:

- *Smistamento*: categorizzazione automatica delle richieste per l'assegnazione (tramite inoltro email o creazione di un trouble ticket) al team di gestione più indicato in funzione del tipo di richiesta.
- *Prioritizzazione*: definizione della priorità da assegnare alle diverse richieste in funzione dell'analisi semantica delle stesse. Tale task può essere compiuto tramite sentiment analysis oppure tramite la rilevazione della presenza di turpiloquio o minacce di disdetta.
- *Risposte automatiche*: comprensione degli intenti espressi dai clienti e invio di risposte immediate e predefinite per ridurre il carico delle richieste agli operatori e velocizzare l'assistenza.
- *Conversation Analysis*: analisi del contenuto delle conversazioni per rilevare esigenze emergenti e avviare azioni di ottimizzazione delle capacità di comprensione dell'Intelligenza Artificiale.
- *Conversation Mining*: possibilità di analizzare le interazioni con i clienti in modo da evidenziare aspetti rilevanti come le aree di insoddisfazione e quelle di soddisfazione, esigenze inespresse, interesse per determinati prodotti.

Anche i dati raccolti dalle interazioni dei clienti possono essere utilizzati dai chatbot per offrire promozioni rilevanti o prodotti e servizi in linea con le esigenze del cliente. Inoltre, i chatbot permettono di interrompere e riprendere la conversazione in qualsiasi momento, senza la necessità di dover richiamare il call-center ed esporre nuovamente il problema da risolvere.

In particolare, il chatbot, se interrogato su problematiche relative ai pagamenti, deve essere in grado di fornire una risposta circa la problematica che ha causato il blocco del pagamento. Anche in questo contesto, sono state individuate due componenti specifiche caratterizzate dalla presenza di modelli di AI: il chatbot e un modello di classificazione multiclasse. L'interazione generale è mostrata in Figura 3.3.

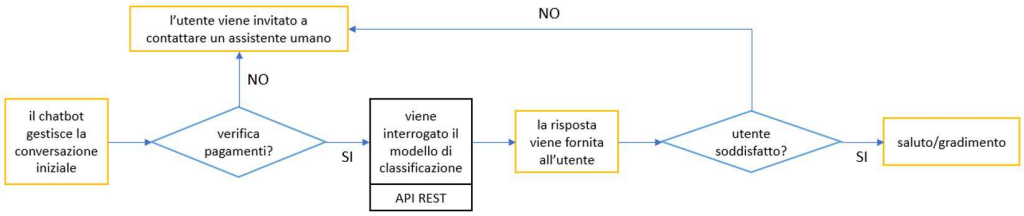


**Figura 3.3:** Flusso conversazionale atteso

La prima interazione con l'utente avverrà attraverso l'interfaccia del chatbot. Da specifiche il chatbot dovrà poter gestire solo casistiche legate alle problematiche di pagamento, quindi se dopo una prima fase di interazione viene individuato un intento che il chatbot non è configurato per gestire (*fallback_intent*) l'utente viene invitato a rivolgersi ad un assistente umano. Se invece si rileva l'intento di verifica viene interrogato tramite API il modello di classificazione addestrato, questa fase è detta di inferenza. Il risultato ottenuto dalla fase di inferenza viene poi utilizzato per fornire la risposta corretta all'utente. A questo punto il chatbot dovrebbe gestire le fasi finali della conversazione, cioè la verifica se la risposta è stata esauriente. In caso negativo, l'assistente rimanda ad un agente umano; in caso positivo esso saluta e chiede un feedback sulla conversazione.

CAPITOLO 4

# Descrizione del sistema complessivo

*Dall'analisi dei requisiti sono emerse le diverse esigenze del cliente. Per soddisfarle si è progettato un sistema composto di diversi moduli con specifiche funzioni. Come anticipato, l'architettura generale della soluzione tecnologica identifica due aree specifiche, una dedicata al recepimento dei dataset e all'ddestramento dei modelli di intelligenza artificiale, l'altra relativa all'esecuzione dei servizi applicativi basati sull'uso dei modelli addestrati. Dal punto di vista infrastrutturale, i due ambiti si differenziano principalmente per la tipologia delle dotazioni hardware necessarie all'implementazione delle funzionalità. Il presente capitolo vuole definire quali sono le componenti che costituiscono il sistema e con che tecnologie sono state implementate.*

## 4.1 Approccio tecnologico

La soluzione proposta vuole rispondere alle esigenze del cliente facendo uso di tecnologie dove l'Intelligenza Artificiale rappresenta la protagonista. I punti di forza sono molteplici. Inanzitutto l'architettura è di tipo *Cloud-based* ed orientata ai microservizi. Infatti, la soluzione si basa su un approccio in cui le componenti alla base vengono sviluppate secondo una logica di container *Docker*, compatibili per un deployment tramite *Kubernetes*. Tale caratteristica rende la soluzione compatabile sia per l'esecuzione nei maggiori Cloud Vendor pubblici, sia per l'esecuzione in cloud privati come quello Filippetti. Come conseguenza, essa è in grado di garantire requisiti di *scalabilità* in base alle esigenze e al carico di lavoro. Infine, tutte le componenti previste per la realizzazione, dettagliate nel seguito del presente capitolo, sono basate sull'adozione di framework e tecnologie di tipo *open source*.

L'approccio tecnologico, come anticipato nel precedente capitolo, consiste di due parti:

- Servizi applicativi finalizzati alla promozione e all'assistenza sui servizi erogati attraverso l'uso di chatbot. In questo caso, gli utenti potranno interagire con il chatbot per esprimere domande in linguaggio naturale; a seguito di ciò, mediante algoritmi di Intelligenza Artificiale, si andrà a identificare il contenuto più adatto per rispondere.
- Servizi applicativi basati sull'uso dell'Intelligenza Artificiale per efficientare le funzionalità del Customer Care. L'obiettivo di questi servizi è quello di ridurre il tempo richiesto agli agenti del Supporto per il riconoscimento e la risoluzione delle problematiche degli utenti (ad esempio, mancati o errati pagamenti per un servizio). Tale servizio è fruibile attraverso il chatbot: gli utenti potranno fornire in linguaggio naturale le informazioni relative alla problematica e l'assistente potrà avviare i controlli automatici al fine di fornire indicazione all'utente circa la possibile problematica e le informazioni operative per la sua risoluzione.

Essendo i due ambiti ben separati, si è deciso di progettare e sviluppare due chatbot distinti, uno per ogni ambito. Tale soluzione ha il vantaggio di garantire la specificità dei dati di training rispetto al dominio in cui ognuno dei due assistenti dovrà operare. Lo switch tra un ambito e l'altro è, comunque, garantito all'utente in qualsiasi momento e verrà gestito tramite specifiche chiamate ad un'API che attivino uno o l'altro chatbot.

Nel seguito sono mostrate le tecnologie di cui il sistema complessivo è costituito, suddivise per ambito.

## 4.2 Ambito Promozionale

L'obiettivo è quello ceare un servizio applicativo utile a migliorare la consultazione dei contenuti digitali disponibili sul sito del cliente relativi ai servizi offerti. Tali servizi sono rivolti agli utenti del sito e prevedono che l'interazione avvenga per mezzo di un assistente virtuale che li aiuti a porre domande sui servizi, identificare lo scopo della domanda, nonchè restituire le informazioni più pertinenti e di interesse per lo specifico utente. Tale interazione è mostrata in Figura 4.1.

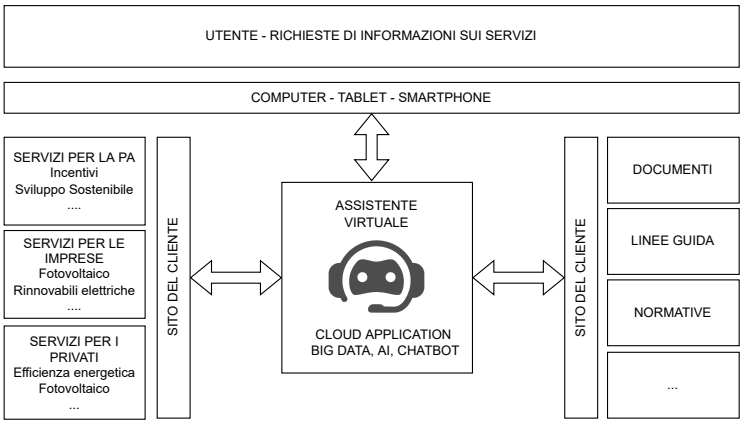


**Figura 4.1:** Richiesta di informazioni su servizi e documenti

Grazie ad un'interazione così strutturata, le idee progettuali potranno essere estese in futuro per riguardare anche le altre tipologie di contentuo, applicando le medesime strategie. L'architettura dei servizi applicativi in risposta all'esigenza è riportata in Figura 4.2.

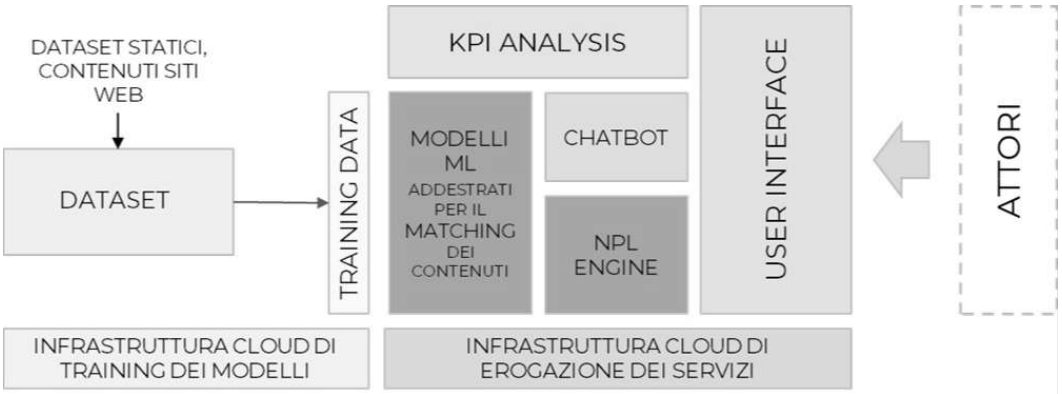


**Figura 4.2:** Soluzione tecnologica

La soluzione prevede due aree distinte: la prima è dedicata all'addestramento dei modelli di Machine Learning, la seconda, invece, all'esposizione dei servizi di consultazione per

gli utenti basati sull'interazione con un assistente virtuale. Gli input per l'addestramento dei modelli derivano dallo scraping del sito web del cliente. Quest'ultimo è stato eseguito tramite la libreria `Selenium` che ha permesso di recuperare, in maniera iterativa su tutte le pagine e sottopagine del sito, le parti di testo strutturate delle pagine HTML e i link a documenti e ad altre aree del sito. Inoltre, per ogni file di testo creato, sono raccolti metadati riguardanti l'URL, il parent URL, il titolo della pagina e il tipo della sorgente (html/pdf). Tali informazioni serviranno per effettuare un primo filtraggio della di base dati.

### 4.2.1 Modello di Neural Search

L'obiettivo del modello è quello di recuperare da una base di dati contenente un vasto quantitativo di documenti di testo il passaggio testuale che meglio corrisponde ad una frase o una domanda specifica. Dal punto di vista tecnico, questa operazione viene svolta convertendo il testo in vettori numerici all'interno di uno spazio vettoriale semantico in cui le parole che esprimono concetti simili sono rappresentate da vettori simili e spazialmente vicini. Per lo sviluppo di questa componente si è fatto uso del framework *Haystack*. Successivamente al crawling del sito del cliente, in Haystack sono stati eseguiti i seguenti step:

- *Creazione di un Document Store*: consiste in un'interfaccia tra i dati e il motore di indicizzazione selezionato (*ElasticSearch*). Una volta istanziato il document store si potrà caricare al suo interno una lista di dizionari ottenibile direttamente a partire dai file prodotti in fase di scraping. In maniera automatica, il document store suddivide i documenti indicizzati in paragrafi.

- *Retriever*: è il primo dei due blocchi "intelligenti" di Haystack. Il suo compito è quello di effettuare una pre-selezione dei paragrafi prodotti dal document store a partire dalla base di dati, rimuovendo i paragrafi che non hanno alcuna pertinenza con la domanda fatta dall'utente. In questa fase è possibile prefiltrare i dati ricorrendo ai metadati che sono stati prodotti durante la fase di scraping. Come mostrato nel capitolo successivo, infatti, nello sviluppo del chatbot è prevista una fase di raccolta di informazioni utili dall'utente per filtrare opportunamente la base di dati.

- *Reader*: è il componente che si occupa dell'analisi strutturale e semantica del linguaggio per recuperare, tra i passaggi selezionati dal retriever, la risposta più appropriata alla domanda. Tale componente si fonda su modelli di deep learning basati su architetture di tipo "Transformer" che funzionano grazie all'uso di modelli pre-addestrati. Le risposte estratte vengono fornite in ordine di punteggio decrescente.

## 4.3 Ambito Customer Care

Le funzionalità previste per supportare l'efficientamento e l'automazione dei servi di Customer Care sono realizzate attraverso componenti informatiche pensate per sfruttare le moderne tecniche basate sull'Intelligenza Artificiale, utili a facilitare l'individuazione delle possibili problematiche relative alle segnalazioni degli utenti. Anche se attualmente il focus della sperimentazione riguarderà problematiche di pagamento, nel futuro le idee progettuali potranno essere estese per affrontare anche le altre possibili problematiche, applicando le medesime strategie. Gli attori coinvolti nel presente ambito sono mostrati in Figura 4.3.

Come mostrato in Figura 4.4, come per l'ambito precedente, l'architettura della soluzione identifica due aree alla base dell'erogazione dei servizi. La prima area, riportata nella parte sinistra della figura, è dedicata alla raccolta dei dataset necessari all'addestramento dei modelli di Machine Learning per il riconoscimento delle possibili problematiche degli

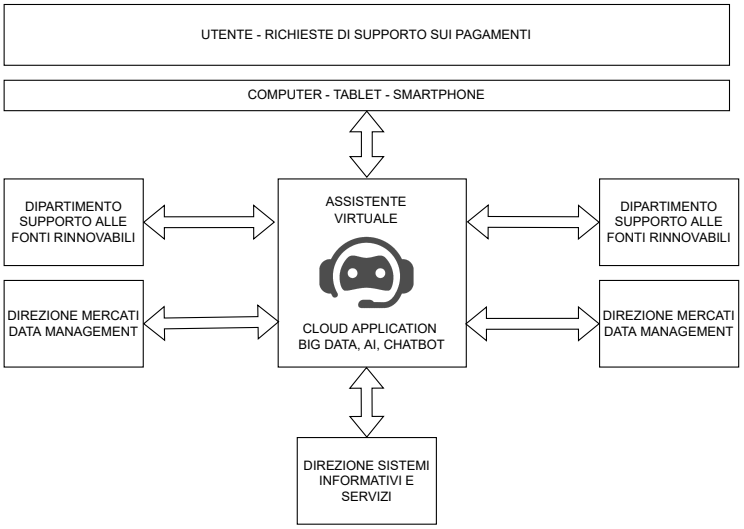


**Figura 4.3:** Attori coinvolti

utenti. Mentre, ad oggi, l'identificazione delle possibili problematiche del servizio per l'utente specifico prevede attività manuali di ricerca e comprensione delle basi di dati da parte dell'Agent di Contact Center, nel futuro, grazie all'uso dell'Intelligenza Artificiale sarà possibile addestrare dei Modelli di Machine Learning per classificare e riconoscere automaticamente la correlazione tra le problematiche maggiormente riscontrate e lo stato degli utenti.

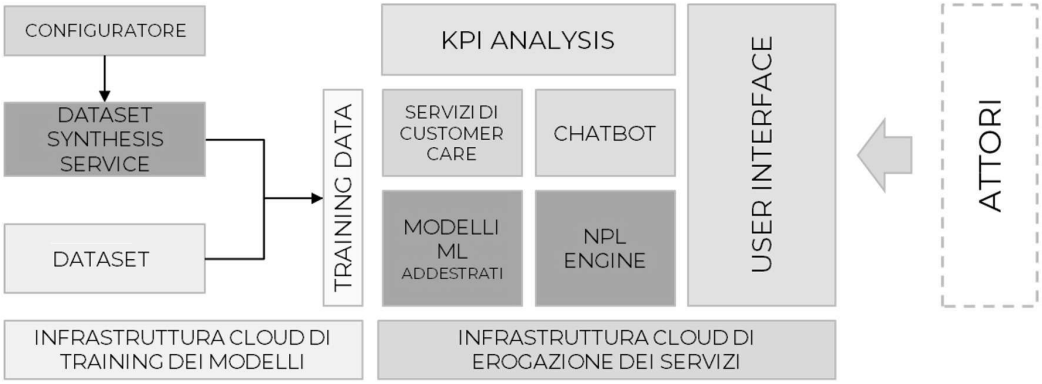


**Figura 4.4:** Soluzione tecnologica

### 4.3.1 Modello di Inferenza

In un documento fornito dal cliente vengono elencati una serie di possibili cause di fallimento di un pagamento. Ad oggi, quando viene inoltrata una richiesta di assistenza, l'agente di contact center provvede manualmente a verificare che il cliente non si trovi in una delle condizioni "bloccanti". L'obiettivo di questo componente è quello di utilizzare la base dati contenente le informazioni relative ai pagamenti (per ogni contratto registrato) con lo scopo rilevare automaticamente le problematiche note. L'ipotesi di partenza è che sia possibile trovare una relazione funzionale tra i dati in possesso e le diverse situazioni bloccanti; l'idea è, quindi, quella di creare un modello supervisionato di classificazione multiclasse. Come schematizzato nella Figura 4.5, una volta individuate le `n` classi che si vogliono identificare e una volta ingegnerizzate le feature appropriate, il modello sarà in grado di estrapolare,

attraverso la fase di addestramento sul dataset fornito, la relazione funzionale `f(x)` che ci permetterà di individuare la problematica a partire dalle feature.

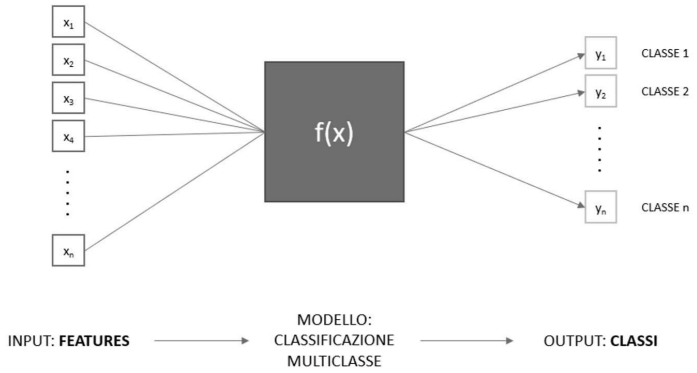


**Figura 4.5:** Modello di inferenza

L'intero sviluppo di questa componente può essere suddiviso concettualmente in 4 blocchi principali, ovvero:

- *Identificazione delle labels*: questa prima parte dello sviluppo è un processo puramente logico/mentale: partendo dalla documentazione fornita, si identificano tutte le possibili cause di blocco e si codifica ognuno degli eventi individuati in una classe definita con un nome sintetico. L'obiettivo finale è quello di razionalizzare il tutto e, quindi, identificare le `n` classi in cui ogni campione del dataset dovrà essere incasellato. Per poter gestire anche situazioni non preventivate, oppure situazioni "regolari", bisognerà prevedere anche queste due ulteriori label.

- *Features Engineering*: partendo dai dati grezzi in nostro possesso, si è in grado di ricavare delle feature che facilitino il lavoro di identificazione dei possibili casi di fallimento. Questo processo consiste in aggregazioni di vario genere (min, max, count, sum, mean, etc.) sui dati forniti. Le feature che ci aspettiamo sono di tipo binario per ognuna delle condizioni di blocco.

- *Labelling del dataset*: una volta individuate le labels target ed le features discriminanti, è necessario associare ad ogni campione del dataset la label corretta. L'algoritmo di labelling è stato pensato come una semplice sequenza di `if-else` dove, a partire dai controlli applicati sui valori di ogni feature, l'algoritmo dovrà assegnare la categoria di problema che più si addice allo stato del campione. Un semplice esempio dell'algoritmo è mostrato nel Listato 4.1.

- *Scelta e addestramento del modello*: il modello scelto è un albero di decisione di tipo *CART*. Il dataset fornito dal cliente presenta un forte sbilanciamento tra le classi individuate; per equilibrare le classi allo stesso livello della classe più presente, sono stati creati dati sintetici tramite over-sampling (*SMOTEN*). L'albero di classificazione è stato allenato sui dati sintetici ed è stato testato con i dati reali, ottenendo ottimi risultati.

```
if stato_contratto == attivo:
    # inizia il controllo delle misure
    if data_ricezione:
        if flg_valida:
            if not fuoripicco:
                exit('Nessun problema')
            ...
else:
    exit('Contratto non attivo')  #label assegnata al campione di input
```

**Listato 4.1:** Esempio dell'algoritmo di labeling

Il prossimo capitolo dettaglierà come il chatbot si interfaccerà con l'utente e con ognuno dei moduli sopra citati.

CAPITOLO 5

# Progettazione del chatbot

*Il presente capitolo vuole delineare come il chatbot dovrà interagire con l'utente per raccogliere le informazioni necessarie ai modelli. In particolare, verranno progettati dei flussi conversazionali più umani possibile. Il principio è quello della semplicità: si vogliono fornire al chatbot le funzionalità essenziali per raggiungere gli obiettivi determinati nella raccolta dei requisiti, garantendo professionalità e chiarezza nelle risposte.*

## 5.1 Sviluppo del chatbot

Nonostante si sia scelto di progettare e implementare due chatbot distinti, uno per ogni ambito, esistono tematiche e argomenti che entrambi i chatbot devono saper affrontare. Nel seguito sono elencati gli intenti e i messaggi che, indipendentemente dall'ambito, l'assistente deve essere in grado di gestire.

- *Messaggio di benvenuto*: quando l'utente clicca sull'icona del chatbot, quest'ultimo deve inviare il primo messaggio. Oltre la presentazione dell'assistente e una breve descrizione delle sue capacità, il chatbot deve permettere all'utente di selezionare l'ambito a cui è interessato

- *Chitchat e FAQ*: entrambi sono casi in cui l'assistente risponde con un set di messaggi predeterminato. Indipendentemente da ciò che l'utente ha scritto precedentemente, il chatbot risponderà sempre allo stesso modo. Questo tipo di interazioni è tipicamente gestito per soddisfare alcune curiosità fuori contesto dell'utente. Alcuni esempi di domande che rientrano in queste categorie sono le seguenti:

  – come ti chiami?
  – chi ti ha programmato?
  – cosa sai fare?
  – etc.

- *Umore negativo*: il chatbot dovrà gestire anche situazioni di scontentezza o rabbia da parte dell'utente. In particolare, quando viene rilevato un intento di tipo "sfida" o "offesa", l'assistente dovrà rimandare l'utente ad un agente umano.

- *Fallback*: mappare qualsiasi possibile domanda dell'utente in intenti non è fattibile. Quando l'utente effettua una richiesta a cui il chatbot non può rispondere (o devia significativamente dai percorsi convenzionali prestabiliti), il chatbot dovrà chiedere di

riformulare la domanda. Nel caso in cui il fallback si verifichi consecutivamente più volte, l'assistente dovrà indirizzare l'utente verso un assistente umano.

- *Feedback*: infine, bisognerà essere in grado di intercettare un feedback dell'utente, che potrà essere positivo o negativo. Il feedback potrebbe essere raccolto in maniera nascosta, effettuando sentiment analysis sull'intera conversazione una volta conclusa. Per alleggerire il carico computazionale dell'intero sistema, però, si è deciso di chiedere esplicitamente all'utente un parere sull'andamento della conversazione e raccoglierne la risposta.

## 5.2 Flusso Conversazionale

In questa sezione verranno definite i flussi conversazionali ideali. In entrambi gli ambiti, infatti, l'utente dovrà fornire alcune informazioni per ricevere una risposta ottimale dai modelli nel *back-end*. Sarà quindi necessario guidare l'utente affinchè provveda nel dare i dati richiesti. Per rendere il fruitore del servizio cosciente di tale necessità, si è deciso di fare uso di pulsanti per l'intero flusso conversazionale. Il loro impiego, infatti, impedisce errori nel processo di *Natural Language Processing* (NLU) e rende consapevole l'utente delle sue possibilità. Allo stesso tempo, è necessario gestire anche le situazioni in cui l'utente non voglia fornire i dati necessari, insistendo se tali dati sono obbligatori per compiere il task richiesto (per esempio, il codice del contratto nell'ambito customer care), o avvertertendolo nel caso in cui tali informazioni abbiano l'unico scopo di garantire risultati più opportuni (per esempio, i metadati per migliorare la ricerca di *Haystack*).

### 5.2.1 Ambito promozionale

In Figura 5.1 è schematizzato il flusso conversazionale previsto nell'ambito promozionale. Per una migliore comprensione, in essa sono stati inseriti dei numeri incrementali per poter dettagliare meglio ogni passaggio. Tali numeri si riferiscono al seguente elenco numerato:

1. L'utente ha cliccato nell'icona del chatbot e riceve un primo messaggio dove l'assistente si presenta e spiega brevemente le proprie capacità. Tramite due pulsanti, l'utente può scegliere quale ambito vuole approfondire. L'ambito promozionale è identificato dal pulsante *Servizi*.

2. Essendo i servizi di diverse tipologie, il chatbot (sempre attraverso dei pulsanti) chiede all'utente se è interessato ai simulatori, agli interventi o ai servizi veri e propri dell'azienda. Per una questione di leggibilità, nella figura sono mostrati soltanto i servizi. Questa scelta rappresenta la prima informazione che potrà essere utilizzata come metadato nella Neural Search.

3. A questo punto, per poter mostrare i servizi dedicati alla propria categoria, viene richiesto all'utente se rappresenta un privato, un'impresa o una Pubblica Amministrazione. Infatti, ogni tipologia di utente ha diversi servizi, interventi e simulatori specifici.

4. Ora il chatbot, tramite dei pulsanti, mostra un elenco dei servizi offerti dall'azienda per la categoria di utente. Quello che l'utente seleziona sarà memorizzato in uno *slot* per poter essere utilizzato nel migliorare la ricerca in *Haystack*.

5. A seguito della selezione del servizio, il chatbot ne fornisce una breve descrizione. Inoltre, tramite un collegamento ipertestuale, rimanda l'utente alla pagina del sito web del cliente dedicata al servizio. Infine, viene proposto all'utente di inserire (nel caso di

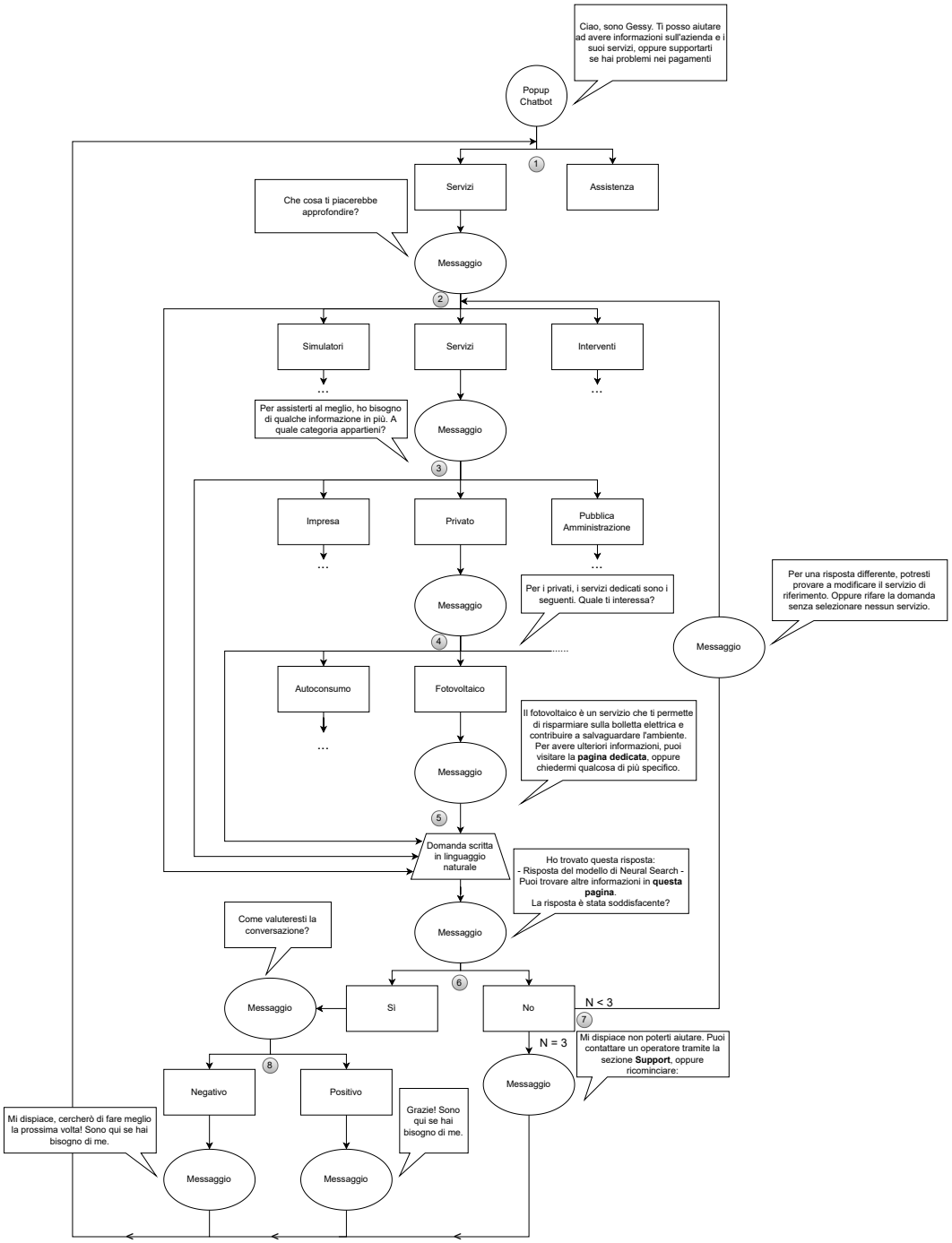


**Figura 5.1:** Flusso conversazionale relativo all'ambito promozionale

una curiosità specifica) nella casella di testo dell'assistente un messaggio in linguaggio naturale. La casella di testo, in realtà, è sempre attiva. Infatti, una volta selezionato l'ambito promozionale, l'utente può in qualsiasi momento porre una domanda in linguaggio naturale da sottoporre al modello di Neural Search, indipendentemente dal numero di metadati raccolti.

6. Il modello restituisce una risposta, l'assistente la elabora e la restituisce all'utente. Oltre alla risposta, viene indicata anche la pagina web o il PDF da cui l'informazione è stata estratta. Il chatbot chiede, infine, un parere sulla qualità della risposta.

7. Nel caso di un feedback negativo, si permette all'utente di modificare il servizio selezionato e di porre nuovamente una domanda. Infatti, i metadati raccolti influenzano fortemente la risposta che il modello fornisce. Tramite un contatore, sarà necessario tenere conto del numero di volte in cui l'utente da un feedback negativo sulla risposta. Al terzo tentativo, l'assistente consiglia di rivolgersi al supporto dell'azienda (fornendo un collegamento ipertestuale) e ricomincia la conversazione.

8. Se, invece, la risposta del modello è stata giudicata positivamente, si chiede un feedback sull'intera conversazione. Infine, si mostrerà un messaggio di saluto dipendente dal tipo di feedback fornito e si permette all'utente, di nuovo, di opzionare l'ambito tramite i pulsanti di inizio conversazione.

Tale flusso dovrà essere implementato principalmente tramite le regole di RASA. In tal modo, in qualsiasi punto della conversazione, l'utente potrà scorrere la chat verso l'alto e selezionare un pulsante diverso da quello selezionato in precedenza. Per esempio, l'utente potrà tornare all'inzio della conversazione e passare dall'ambito promozionale a quello customer care opzionando il pulsante *Assistenza*.

### 5.2.2 Ambito customer care

In Figura 5.2 è rappresentato il flusso conversazionale per l'ambito customer care. Come per l'ambito promozionale, i numeri in figura si riferiscono al seguente elenco:

1. Il messaggio iniziale è, ovviamente, lo stesso rispetto all'ambito promozionale. Ma per esplorare le funzionalità dell'ambito customer care, l'utente deve selezionare il pulsante *Assistenza*.

2. L'assistente controlla se l'utente è autenticato (condizione necessaria per il prosieguo) e, nel caso, inizia a chiamarlo con il suo nome per garantire un'esperienza più "human-like". Se, invece, l'utente non è autenticato nel sito del cliente, si dettagliano i passaggi per effettuare il login.

3. Selezionando il pulsante *Altro*, l'assistente avverte che tale funzione non è ancora implementata. Sviluppi futuri prevedono di integrare *Haystack* anche in questo ambito per permettere di porre domande relative alla modulistica in linguaggio naturale. Attualmente si rimanda l'utente alla pagina del supporto.

4. Opzionando il pulsante *Pagamenti*, invece, l'assistente fornisce il codice della pratica (o delle pratiche) attiva nell'account. L'utente deve confermare che il codice della pratica corrisponde, nel caso in cui ne abbia soltanto una attiva, oppure selezionare quale pratica vuole approfondire, nel caso ne abbia più di una.

5. A questo punto, si chiede se l'utente vuole maggiori informazioni relative alla pratica opzionata o se vuole verificare la presenza di problematiche nei pagamenti.

6. Nel primo caso, si riassumono in un messaggio i dati principali della pratica, quali lo stato del contratto, la data di attivazione, il codice POD, etc. Si permette, poi, all'utente di contattare il supporto o di individuare le problematiche nel contratto.

7. Nel secondo caso, l'assistente deve elaborare i risultati del modello di inferenza, tradurli in linguaggio naturale e restituirli all'utente. Il modello addestrato segnala la presenza o l'assenza di problemi nella pratica opzionata precedentemente e l'utente potrà, a questo punto, procedere con una delle seguenti scelte:

- Ritornare alla lista delle pratiche. Nel caso in cui un utente abbia più pratiche attive, potrebbe volerne verificare l'integrità.
- Confermare la risoluzione dei dubbi.
- Inviare una segnalazione.
- Dettagliare le informazioni relative alla pratica.

8. Se l'utente decide di inviare una segnalazione, gli è richiesto di scrivere nella casella di testo il messaggio che vuole inoltrare al supporto. A seguito della conferma della ricezione della segnalazione da parte dell'assistente, si rimanda l'utente alla scelta iniziale tra i due ambiti.

9. Se l'utente, invece, conferma l'utilità della soluzione proposta dal chatbot in fase di inferenza, gli viene richiesto un feedback sull'intera conversazione.

10. A seconda del tipo di feedback fornito, il chatbot saluta e rimanda alla scelta iniziale tra i due ambiti.

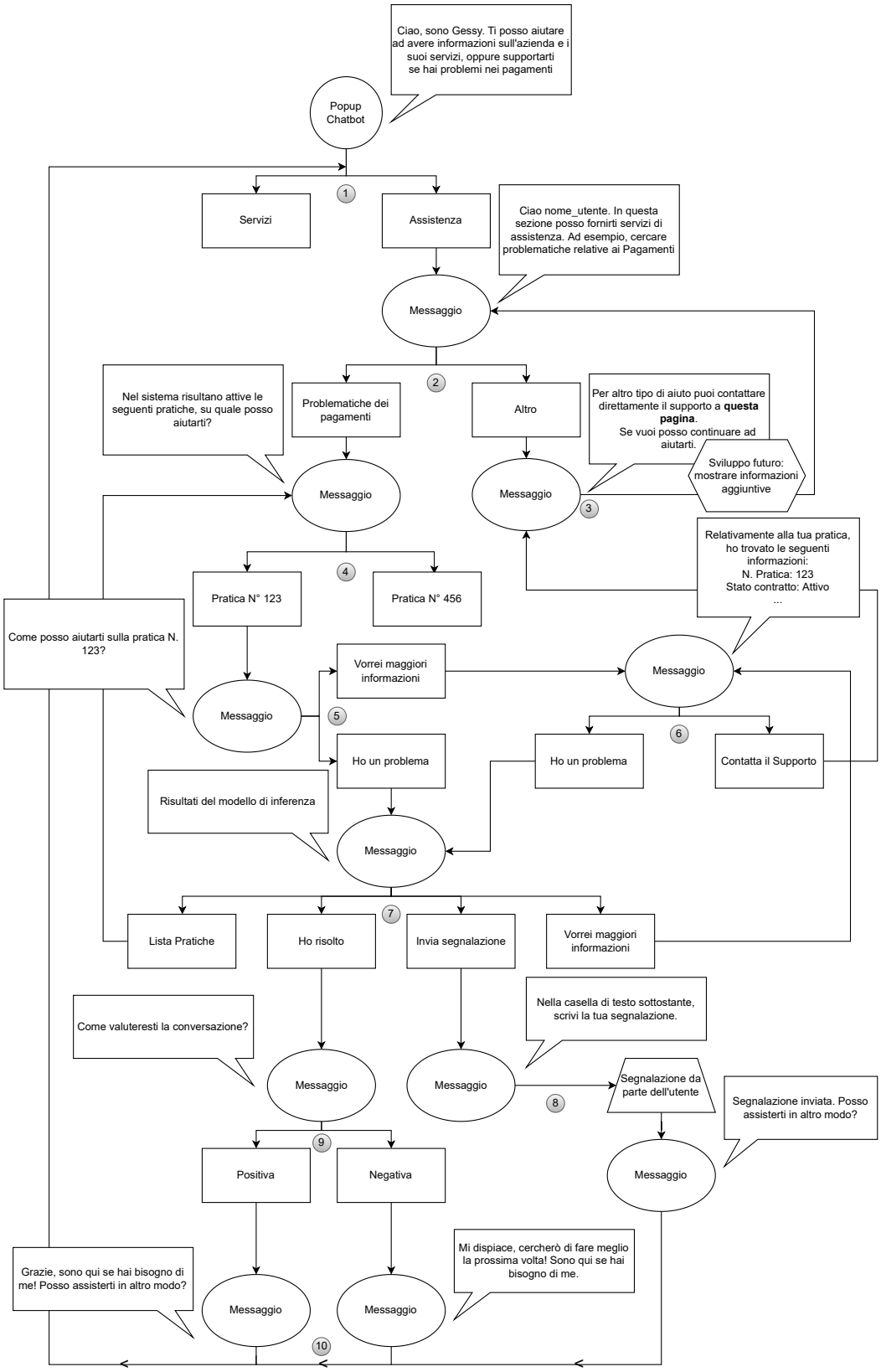


**Figura 5.2:** Flusso conversazionale relativo all'ambito customer care

# CAPITOLO 6

# Implementazione

*Avendo ben chiari i flussi conversazionali che si vogliono ottenere, in accordo a quanto definito nella Sezione 2.2, il presente capitolo mostra l'implementazione dei file di configurazione per ogni ambito. Per una migliore comprensione, i contenuti dei file non sono mostrati nella loro interezza ma seguendo i flussi conversazionali definiti nel capitolo precedente.*

## 6.1 Ambito promozionale

Come anticipato nel precedente capitolo, per permettere all'utente di premere qualsiasi pulsante comparso fino a quel momento nella conversazione, non si è fatto uso del file `stories.yml`. Il Core di RASA sarà, quindi, addestrato soltanto con un insieme di regole. Procedendo con ordine, di seguito viene esposto il contenuto dei file principali.

### 6.1.1 Config

Questo file ha un duplice scopo: definire la Pipeline per l'esecuzione del *Natural Language Understanding* (NLU) e istanziare le *Dialogue Policies*. Nel Listato 6.1 sono elencati i componenti che costituiscono la pipeline di NLU.

```
pipeline:
  - name: WhitespaceTokenizer
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: "word"
    min_ngram: 1
    max_ngram: 1
    OOV_token: "_oov_"
    use_shared_vocab: False
  - name: DIETClassifier
    transformer_size: 256
    number_of_transformer_layers: 2
    epochs: 300
    random_seed: 123
    learning_rate: 0.001
    embedding_dimension: 20
    entity_recognition: False
    constrain_similarities: True
  - name: ResponseSelector
```

```
    epochs: 300
    constrain_similarities: True
  - name: FallbackClassifier
    threshold: 0.7
    ambiguity_threshold: 0.1
```

**Listato 6.1:** Pipeline di NLU

Si vanno ora a dettagliare le caratteristiche di ogni componente e spiegare i motivi che hanno condotto alla scelta delle loro configurazioni:

- `WhiteSpaceTokenizer`: crea un token per ogni spazio bianco che separa una sequenza di caratteri.
- `LexicalSyntacticFeaturizer`: crea le feature per l'estrazione delle entità. Scorre una finestra mobile ogni token del messaggio dell'utente e crea feature in base alla configurazione.
- `CountVectorsFeaturizer`: crea feature per la classificazione degli intenti e la selezione delle risposte. Di particolare interesse è il parametro `OOV_token` impostato a `"_oov_"`. "OOV" è l'acronimo di "Out Of Vocabulary"; non avendo incluso un dizionario pre-addestrato (per esempio, *Spacy*), tutto ciò che non è compreso nei dati di training della pipeline non è conosciuto dal modello di NLU. Il token `_oov_` indica, quindi, una parola di cui non si conosce il significato. Tale impostazione si renderà utile per capire quando sarà necessario passare al modello di Neural Search la domanda posta dall'utente.
- `DIETClassifier`: con "DIET si intende "Dual Intent and Entity Transformer", tale componente compie un duplice task, ovvero la classificazione degli intenti e l'estrazione delle entità.
- `ResponseSelector`: predice una risposta del chatbot a partire da un set di risposte. Tale predizione è usata dal *Dialogue Manager* per emettere la risposta predetta.
- `FallbackClassifier`: classifica un messaggio con l'intento `nlu_fallback` se la confidenza della classificazione dell'intento del `DIETClassifier` non supera il 70%.

Per quanto riguarda le *Dialogue Policies*, mostrate nel Listato 6.2, queste non differiscono dalla configurazione standard. Il loro funzionamento è stato precedentemente descritto nella Sezione 2.2.3.

```
policies:
  - name: MemoizationPolicy
  - name: UnexpecTEDIntentPolicy
    max_history: 5
    epochs: 300
    constrain_similarities: True
  - name: RulePolicy
```

**Listato 6.2:** Dialogue Policies

### 6.1.2 Domain e NLU

Nel file di `Domain.yml` vengono definiti i nomi degli intenti, delle entità e delle custom actions. Inoltre, vengono istanziati gli slot e vengono espresse le risposte che non richiedono codice Python per la loro esecuzione. Nel file di `NLU.yml`, invece, sono elencati molteplici

esempi per ogni intento. Nel Listato 6.3 sono elencati gli intenti necessari per l'esecuzione di questo ambito e le possibili entità estraibili.

```
intents:
- out_of_scope
- saluto
- chitchat
- negazione
- grazie
- addio_pos
- addio_neg
- affermazione
- nlu_fallback
- esplorazione_servizi
- dichiarazione_stato
- info_servizi
- cosa_approfondire
- complimento
entities:
- scope
- servizi
- soggetto_giuridico
- tipo_servizi
- slot_was_set
- feedback
```

**Listato 6.3:** Definizione degli intenti e delle entità

Per alcuni degli intenti, nel file di `NLU.ym`, sono forniti diversi esempi, mostrati nel listato 6.4.

```
nlu:
  - intent: out_of_scope
    examples: |
      - come si fa _oov_ per avere accesso a _oov_?
      - vorrei sapere se e' possibile _oov_
      - cosa si intende per _oov_?
      - come faccio a _oov_?
      - che cos'e' il _oov_?
      - come viene _oov_ per _oov_?
      - in cosa consiste _oov_?
      - cosa prevede _oov_?
  - intent: chitchat/scherzo
    examples: |
      - posso ordinare una pizza?
      - quanti anni hai?
      - raccontami una barzelletta
      - chi e' il presidente degli stati uniti?
      - quanto fa 2+2?
      - cosa mangio a cena?
      - ti piace _oov_?
      - qual e' il tuo hobby?
  - intent: chitchat/area_clienti
    examples: |
      - come si accede all'area clienti?
      - come faccio ad entrare nell'area clienti?
      - devo entrare nell'area clienti
      - devo accedere
      - come si accede?
```

```
    - area clienti
    - avrei bisogno di accedere all'area clienti
    - come si accede all'area clienti?
- intent: chitchat/come_stai
  examples: |
    - come stai?
    - come va?
    - che mi racconti?
    - cosa mi dici?
    - come te la passi?
- intent: chitchat/capacita
  examples: |
    - cosa sai fare?
    - come puoi aiutarmi?
    - a cosa servi?
    - capacita'
    - cosa puoi fare?
    - in cosa puoi assistermi?
    - quali sono le tue capacita'?
    - cosa fai?
- intent: chitchat/bot_challenge
  examples: |
    - sei un bot?
    - sei un umano?
    - sto parlando con un robot?
    - sto parlando con un uomo?
    - con chi sto parlando?
    - chi sei?
    - come ti chiami?
    - qual e' il tuo nome?
- intent: addio_pos
  examples: |
    - arrivederci
    - addio
    - buona giornata
    - a dopo
    - a piu' tardi
    - a presto
    - ciao ciao
    - alla prossima
- intent: grazie
  examples: |
    - grazie
    - grazie mille
    - molto gentile
    - ti ringrazio
    - sei stato molto utile
    - thanks
    - grazie per l'aiuto!
    - e' stato un piacere
- intent: negazione
  examples: |
    - no
    - n
    - mai
    - penso di no
    - non credo
```

```
    - non proprio
    - non molto
    - per niente
    - non ci penso
- intent: affermazione
  examples: |
    - ok
    - va bene
    - certo
    - si
    - yes
    - certamente
    - d'accordo
    - abbastanza
- intent: ricominciare
  examples: |
    - ricominciamo?
    - voglio riniziare
    - voglio ricominciare la conversazione
    - riniziare
    - restart
    - yes
    - certamente
    - d'accordo
    - abbastanza
```

**Listato 6.4:** Contenuto del file `nlu.yml`

L'attenzione può essere rivolta nel primo intento dove è presente il token `_oov_`. Qualsiasi parola non nominata nel file `nlu.yml` verrà corrisposta a questo token. In tal modo, è stato possibile non includere gli ipoteticamente infiniti esempi di domande che l'utente potrebbe porre sul contenuto del sito web dell'azienda. Un'altra nota è sull'intento `Chitchat`, definito di tipo `retrieval`. Ciò ha permesso la sua suddivisione in sotto-intenti distinti; nonostante tutti gli esempi si ritrovino nella categoria "chiacchere", essi possono essere divisi in categorie separate, così da assegnare una risposta specifica per ogni tipologia di chiacchera individuata. Infine, si evidenzia come a non tutti gli intenti definiti nel `Domain` corrispondono degli esempi; alcuni intenti, infatti, saranno adoperati esclusivamente come argomento dei pulsanti.

Ritornando al file di `Domain`, vengono anche definiti quattro slot e tre custom action, la cui istanziazione è mostrata nel Listato 6.5. Gli slot `tipo_servizi` e `servizi` indicano rispettivamente la tipologia di servizio che l'utente sta approfondendo (interventi, simulatori o servizi veri e propri) e a quale servizio specifico sta ponendo l'attenzione (autoconsumo, fotovoltaico, etc.). Entrambi gli slot saranno utili come metadati per migliorare la risposta del modello di Neural Search. Lo slot `soggetto_giuridico`, invece, memorizza se l'utente dichiara di essere un privato, un'impresa o un rappresentante di una Pubblica Amministrazione. Tale informazione è utile per mostrare i servizi, gli interventi o i simulatori dedicati ad ogni categoria. Essendo un dato statico, salvarlo in uno slot permette di non doverlo richiedere ogni volta che l'utente richiede la lista di servizi dedicati. Infine, lo slot `contatore` serve a numerare le domande rivolte ad *Haystack* la cui risposta non ha soddisfatto l'utente. Infatti, come previsto dal flusso conversazionale descritto nella Sezione 5.2.1, se l'utente mostra insoddisfazione nelle risposte per tre volte di fila, l'assistente lo indirizza verso un agente umano.

```
slots:
  tipo_servizi:
    type: rasa.shared.core.slots.TextSlot
    initial_value: null
```

```
    auto_fill: true
    influence_conversation: false
  soggetto_giuridico:
    type: rasa.shared.core.slots.TextSlot
    initial_value: null
    auto_fill: true
    influence_conversation: false
  servizi:
    type: rasa.shared.core.slots.TextSlot
    initial_value: null
    auto_fill: true
    influence_conversation: false
  contatore:
    type: rasa.shared.core.slots.FloatSlot
    initial_value: 0
    auto_fill: false
    influence_conversation: false
    max_value: 3.0
    min_value: 0.0

actions:
- haystack
- info_generiche_servizio
- ask_servizi
- ask_soggetto_giuridico
```

**Listato 6.5:** Elenco degli slot

Per quanto riguarda le risposte, nonostante queste siano contenute nel file di *Domain*, verranno dettagliate nella prossima sezione per facilitare la comprensione del flusso conversazionale.

### 6.1.3 Rules, Responses e Custom Actions

In accordo con quanto definito nella Sezione 5.2.1, la conversazione inizia con un messaggio di benvenuto. Tale messaggio è innescato dall'interfaccia web che, nel momento in cui l'utente clicca nell'icona del chatbot, invia un l'intento `saluto` a RASA. Questa prima interazione è gestita dalla regola mostrata nel Listato 6.6.

```
- rule: rispondi con messaggio benvenuto
  conversation_start: true
  steps:
    - intent: saluto
    - action: utter_benvenuto
```

**Listato 6.6:** Regola di benvenuto

La risposta `utter_benvenuto`, definita nel *Domain*, è riportata nel Listato 6.7.

```
utter_benvenuto:
- buttons:
  - payload: /cosa_approfondire{"scope":"on"}
    title: Servizi
  - payload: /cosa_approfondire{"scope":""}
    title: Assistenza
  text: Ciao sono Gessy, l'assistente virtuale di XXX.
  Ti posso aiutare ad avere informazioni su XXX,
  i suoi servizi, oppure supportarti se hai problemi nei pagamenti.
```

**Listato 6.7:** Messaggio di benvenuto

Tramite i pulsanti definiti al di sotto della parola chiave `buttons` si permette all'utente di selezionare l'ambito. Si evidenzia come il payload dei pulsanti è costituito dallo stesso intento, ma l'entità `scope` viene valorizzata solamente per l'ambito promozionale. Tale entità verrà controllata dal front-end e attiverà il chatbot corrispondente. Per motivi di riservatezza, il nome dell'azienda cliente rimane anonimo nel presente lavoro.

La regola definita nel Listato 6.8 fa procedere la conversazione intercettando l'intento `cosa_approfondire` e mandando in output il contenuto della risposta `utter_ask_tipo_servizi`, mostrata nello stesso listato.

```
  - rule: chiedere tipo servizi
    steps:
      - intent: cosa_approfondire
      - action: utter_ask_tipo_servizi

utter_ask_tipo_servizi:
- buttons:
- payload: /esplorazione_servizi{{"tipo_servizi":"servizi"}}
  title: Servizi
- payload: /esplorazione_servizi{{"tipo_servizi":"interventi"}}
  title: Interventi
- payload: /esplorazione_servizi{{"tipo_servizi":"simulatori"}}
  title: Simulatori
text: Che cosa ti piacerebbe approfondire?
```

**Listato 6.8:** Regola per la selezione della tipologia del servizio con conseguente risposta

Di nuovo, l'intento dei tre payload è il medesimo. È l'entità `tipo_servizi`, che verrà memorizzata nell'omonimo slot, che definisce la tipologia di servizio selezionata.

La regola nel Listato 6.9 consente la continuazione della conversazione.

```
  - rule: richiesta soggetto giuridico
    steps:
      - intent: esplorazione_servizi
      - action: ask_soggetto_giuridico
```

**Listato 6.9:** Regola per la richiesta della tipologia di utente

L'azione `ask_soggetto_giuridico` è la prima custom action ad essere eseguita. La sua implementazione è mostrata nel Listato 6.10.

```
class ask_soggetto_giuridico(Action):
    def name(self) -> Text:
        return "ask_soggetto_giuridico"

    def run(self, dispatcher: CollectingDispatcher, tracker: Tracker,
        ↪ domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        ruolo = tracker.get_slot('soggetto_giuridico')
        button = []
        if ruolo == None:
            for counter, el in enumerate(servizi.keys()):
                if counter < 3:
                    payload = "/dichiarazione_stato{\"soggetto_giuridico
                        ↪ \":\"" + el.lower() + "\"}"
                    el = el.capitalize()
                    if el == "Pa":
                        el = "PA"
                    button.append({"title": el.capitalize(), "payload":
                        ↪ payload})
```

```
            else: break
        dispatcher.utter_message("Per assisterti al meglio, ho
            ↪ bisogno di qualche informazione in piu'. A quale
            ↪ categoria appartieni?", buttons = button)
        return []
    else:
        return [rasa_sdk.events.FollowupAction("ask_servizi")]
```

**Listato 6.10:** Custom action per la richiesta della tipologia dell'utente

Tale azione effettua inizialmente un controllo: se l'utente ha dichiarato precedentemente la sua categoria, allora tramite l'evento `FollowupAction` viene richiamata la funzione `ask_servizi`, dettagliata nel seguito. Al contrario, se il flusso è alla prima iterazione, vengono estratte da un dizionario le chiavi che costituiscono le possibili tipologie dell'utente. Tali chiavi andranno a comporre i testi dei pulsanti, il cui payload è costituito dall'intento `dichiarazione_stato`.

Tale intento attiva la regola mostrata nel Listato 6.11.

```
- rule: selezionare servizio
  steps:
   - intent: dichiarazione_stato
   - action: ask_servizi
```

**Listato 6.11:** Regola per la selezione della tipologia del servizio

L'azione `ask_servizi` permette all'utente di selezionare il servizio, l'intervento o il simulatore che vuole approfondire. Per motivi di estensione del codice, esso non viene incluso per intero nel presente lavoro. Oltre alla selezione del servizio dedicato alla propria categoria, in questa azione viene effettuato il controllo sul contatore che numera le risposte dal modello di Neural Search che non hanno soddisfatto l'utente. Tale controllo è mostrato nel Listato 6.12.

```
contatore = tracker.get_slot('contatore')
if contatore > 2:
    button = [{"title": 'Servizi', "payload": '/cosa_approfondire{{"scope
        ↪ ": "on"}}'}, {"title": 'Assistenza', "payload": '/
        ↪ cosa_approfondire{{"scope": ""}}'}]
    dispatcher.utter_message("Mi dispiace non poterti aiutare. Puoi
        ↪ contattare un operatore tramite la sezione <a href='https://XXX
        ↪ 'target='_blank'>Support</a>, oppure ricominciare:", buttons =
        ↪ button)
    return [SlotSet("contatore", 0)]
if tracker.latest_message['intent'].get('name') == 'negazione':
    contatore = contatore + 1
```

**Listato 6.12:** Controllo del contatore

Nel seguito della funzione si generano dei pulsanti con l'elenco degli interventi, dei simulatori e dei servizi dedicati. Il payload di tali pulsanti è composto dall'intento `info_servizi` oltre che dalla specifica entità che andrà a valorizzare lo slot *servizi* con il servizio, l'intervento o il simulatore opzionato.

La regola nel Listato 6.13 attiva la custom action `info_generiche_servizio`. Tale funzione, per ogni servizio, intervento o simulatore ne fornisce una breve descrizione e la pagina del sito web dove se ne parla più approfonditamente. L'assistente richiede all'utente, successivamente, di domandare qualcosa di più specifico.

```
contatore = tracker.get_slot('contatore')
- rule: rispondi quando viene selezionato un bottone servizio
steps:
```

```
  - intent: info_servizi
  - action: info_generiche_servizio
```

**Listato 6.13:** Regola per generare informazioni sui servizi

Se l'utente scrive qualcosa, deve intervenire il modello di Neural Search. Per far ciò sono definite due regole, mostrate nel Listato 6.14.

```
contatore = tracker.get_slot('contatore')
  - rule: quando la confidenza NLU e' bassa interroga haystack
    steps:
      - intent: nlu_fallback
      - action: haystack
  - rule: se out of scope interroga haystack
    steps:
      - intent: out_of_scope
      - action: haystack
```

**Listato 6.14:** Regole per l'interrogazione del modello

La prima esegue la custom action di nome "haystack" quando la confidenza sulla classificazione degli intenti non supera la soglia definita nella pipeline di NLU. La seconda esegue la stessa custom action quando viene identificato l'intento "out_of_scope" che, come visto nella precedente sezione, fa largo uso del token _oov_ (Out Of Vocabulary).

L'implementazione della custom action `haystack` è descritta nel Listato 6.15. Nonostante la sua estensione, si è deciso di riportarla nella sua interezza perché rappresenta il cuore dell'ambito promozionale.

```
class Haystack(Action):
    def name(self) -> Text:
        return "haystack"

    def get_haystack(self):
        url = os.getenv("HAYSTACK_QUERY_API")
        return url if url is not None else "http://localhost:8001/query"

    def get_confidence_threshold(self):
        confidence = os.getenv("HAYSTACK_CONFIDENCE_THRESHOLD")
        return float(confidence) if confidence is not None else 0.7

    def get_max_k_retriever(self):
        kretriever = os.getenv("HAYSTACK_MAX_K_RETRIEVER")
        return int(kretriever) if kretriever is not None else 5

    @staticmethod
    def get_first_answer(response):
        if response is None or len(response["answers"]) == 0:
            return None
        else:
            return response["answers"][0]

    def get_answer_from_context(self, res_context: str):
        list = []
        list1 = []
        if (res_context) is None:
            return None
        if len(res_context) < 160:
            return res_context
        for counter, el in enumerate(res_context):
```

```
            if counter <160:
                list.append(el)
            else: list1.append(el)
        if len(list1) > 0:
            context = '{} <span class="haystack_full_context" style="
                ↪ display:none"> {}</span>'.format(listToString(list),
                ↪ listToString(list1))
        else: context =listToString(list)
        return context

    def run(self, dispatcher: CollectingDispatcher, tracker: Tracker,
        ↪ domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
        _input = tracker.latest_message['text']
        tipo_servizi = tracker.get_slot('tipo_servizi')
        servizio = tracker.get_slot('servizi')
        if servizio != None:
            servizio.replace(' ', '-')
        if tipo_servizi == 'interventi' or tipo_servizi == 'simulatori':
            payload = {"query": str(_input), "params": {"Retriever": {"
                ↪ top_k": self.get_max_k_retriever()}, "filters": {
                "domain": [str("interventi-e-simulatori"), "manuali-
                    ↪ moduli-e-procedure"]}}}
        elif tracker.get_slot('servizi') == None:
            payload = {"query": str(_input), "params": {"Retriever": {"
                ↪ top_k": self.get_max_k_retriever()}}}
        elif tracker.get_slot('ruolo') == 'pa':
            payload = {"query": str(_input), "params": {"Retriever": {"
                ↪ top_k": self.get_max_k_retriever()}, "filters": {
                "domain": ["pa/" + str(servizio), "manuali-moduli-e-
                    ↪ procedure"]}}}
        else:
            payload = {"query": str(_input), "params": {"Retriever": {"
                ↪ top_k": self.get_max_k_retriever()}, "filters": {
                "domain": [str(servizio), "manuali-moduli-e-procedure
                    ↪ "]}}}
        logging.info("Calling haystack. Payload is: " + str(payload))
        url = self.get_haystack()
        headers = {'Content-Type': 'application/json', 'Accept': '
            ↪ application/json'}
        try:
            response = requests.request("POST", url, headers=headers,
                ↪ json=payload).json()
        except:
            print(traceback.format_exc())
            print('Qualcosa e' andato storto')
            dispatcher.utter_message('Qualcosa e' andato storto')
            return []
        answer = self.get_first_answer(response)
        confidence = None
        if answer is not None and "score" in answer:
            confidence = answer["score"]
            logging.info("Got answer with confidence " + str(confidence))
        else:
            logging.warning("No valid answer from haystack. query: " +
                ↪ str(payload))
        if confidence is None or confidence < self.
            ↪ get_confidence_threshold():
```

```
            logging.warning("Too low confidence from haystack. Expected
                ↪ at least {} but given {}"
                            .format(self.get_confidence_threshold(),
                                ↪ confidence if confidence is not None
                                ↪ else "-"))
            button = []
            ruolo = tracker.get_slot('soggetto_giuridico')
            if tipo_servizi=='interventi':
                for el in servizi["interventi"]:
                    payload = "/info_servizi{\"servizi\":\"" + el.lower()
                        ↪ + "\"}"
                    button.append({"title": el, "payload": payload})
            elif tipo_servizi == 'simulatori':
                for el in servizi["simulatori"]:
                    payload = "/info_servizi{\"servizi\":\"" + el.lower()
                        ↪ + "\"}"
                    button.append({"title": el, "payload": payload})
            elif tipo_servizi == 'servizi':
                if ruolo == 'privato':
                    for el in servizi["privato"]:
                        payload = "/info_servizi{\"servizi\":\"" + el.
                            ↪ lower() + "\"}"
                        button.append({"title": el, "payload": payload})
                elif ruolo == 'impresa':
                    for el in servizi["impresa"]:
                        payload = "/info_servizi{\"servizi\":\"" + el.
                            ↪ lower() + "\"}"
                        button.append({"title": el, "payload": payload})
                elif ruolo == 'pa':
                    for el in servizi["pa"]:
                        payload = "/info_servizi{\"servizi\":\"" + el.
                            ↪ lower() + "\"}"
                        button.append({"title": el, "payload": payload})
                else:
                    for el in servizi["privato"]:
                        payload = "/info_servizi{\"servizi\":\"" + el.
                            ↪ lower() + "\"}"
                        button.append({"title": el, "payload": payload})
            else:
                for el in servizi["privato"]:
                    payload = "/info_servizi{\"servizi\":\"" + el.lower()
                        ↪ + "\"}"
                    button.append({"title": el, "payload": payload})

            dispatcher.utter_message("Non riesco a risponderti a questa
                ↪ domanda, prova a riformulare o a selezionare un nuovo
                ↪ servizio", buttons = button)
            print("Low confidence for query '{}'. Got {}".format(str(
                ↪ _input), str(confidence)))
            contatore = tracker.get_slot('contatore')
            return [SlotSet("contatore", contatore+1)]

        text = self.get_answer_from_context(answer["context"]).replace("\
            ↪ n\n","\n")
        if text is None:
            dispatcher.utter_message("Non ho trovato nessuna risposta")
            return []
```

```
        dispatcher.utter_message("Ho trovato questa risposta:")
        dispatcher.utter_message("{}".format(text))
        source = answer["meta"]["url"]
        dispatcher.utter_message(
            "Puoi trovare altre informazioni in questa <a href=" + source
                ↪ + " target='_blank'>pagina</a>")
        # other responses
        other_responses = []
        for i, res in enumerate(response["answers"]):
            if i == 0:
                continue
            if "context" not in res:  # see e.g. "vorrei sapere come
                ↪ istallare dei pannelli fotovoltaici"
                continue

            text = self.get_answer_from_context(res["context"]).replace
                ↪ ("\n\n","\n")
            if text is None:
                continue
            x = {
                "text": "{}".format(text),
                "link": "Puoi trovare altre informazioni in questa <a
                    ↪ href="
                        + res["meta"]["url"] + " target='_blank'>pagina</
                            ↪ a>"
            }
            other_responses.append(x)
        dispatcher.utter_message("hidden:haystack_more", json_message=
            ↪ other_responses)
        button = [{"title": 'So', "payload": '/affermazione'}, {"title":
            ↪ 'No', "payload": '/negazione'}]
        dispatcher.utter_message("La risposta e' stata soddisfacente?",
            ↪ buttons = button)
        return []
```

**Listato 6.15:** Custom action per l'interrogazione ad Haystack

Procedendo con ordine, vengono inizialmente definite delle funzioni che hanno il compito di estrarre delle variabili d'ambiente. In particolare, vengono istanziate le variabili relative all'URL dove è in esecuzione *Haystack*, la soglia di confidenza che la risposta del modello dovrà sormontare e il numero di risposte dal modello che verranno date in output. Successivamente, viene definita la funzione `get_answer_from_context` che ha lo scopo di integrare nel testo della risposta del codice HTML, per mostrare soltanto i primi 160 caratteri, e un pulsante "Vedi altro" per espandere l'intero contenuto. Nel `run` si formattano gli slot memorizzati durante la conversazione affinché siano compatibili con i metadati che si aspetta il modello. Infine, si esegue una richiesta POST ad *Haystack* ed effettuati i controlli sulla confidenza. Nel caso il controllo abbia un esito negativo, il contatore viene incrementato di uno e si richiede all'utente di riformulare la domanda o selezionare un nuovo servizio, intervento o simulatore. Altrimenti, viene mostrata la prima risposta e, integrando del codice HTML, un pulsante che permette all'utente di visualizzare ulteriori quattro risposte che hanno ottenuto un punteggio più basso. Alle risposte viene sempre associato il collegamento ipertestuale alla pagina web o al PDF da cui esse sono state estratte. In ultimo, si richiede all'utente se ha trovato la risposta soddisfacente.

- Nel caso egli dica di sì, verrà predetto l'intento "affermazione". Esso attiva la regola

definita nel Listato 6.16 che, a sua volta, innesca la risposta `utter_feedback`.

```
- rule: feedback
  steps:
    - intent: affermazione
    - action: utter_feedback
```

**Listato 6.16:** Regola per la richiesta di feedback

Il feedback viene raccolto e l'utente viene salutato. Per completare il ciclo, si ripropone all'utente la scelta iniziale tra il pulsante "Servizi" e quello "Assistenza".

- Nel caso l'utente neghi di essere soddisfatto, viene predetta l'azione `ask_servizi` tramite la regola definita nel Listato 6.17. Il contenuto di tale azione è già stato chiarito precedentemente.

```
- rule: chiedere dei servizi
  steps:
    - intent: negazione
    - action: ask_servizi
```

**Listato 6.17:** Regole per la gestione di utenti insoddisfatti

In ultimo, le chiacchere vengono gestite con la regola definita nel Listato 6.18. In definitiva, tutto ciò che l'utente scrive e che non viene classificato con intento `chitchat`, viene passato al modello di Neural Search con gli opportuni controlli sulla confidenza della risposta.

```
- rule: rispondi alle domande chitchat
  steps:
    - intent: chitchat
    - action: utter_chitchat
```

**Listato 6.18:** Regola per la gestione delle chitchat

L'azione `utter_chitchat` fornisce una risposta specifica per ciasscuno degli intenti di tipo `chitchat` definiti precedentemente.

## 6.2 Ambito customer care

Per gli stessi motivi descriti per l'ambito promozionale, il file `stories.yml` non è stato utilizzato per l'implementazione del presente ambito.

### 6.2.1 Custom channel

L'esigenza di riconoscere un utente autenticato ha portato a sfruttare la caratteristica open source del framework. Infatti, RASA si aspetta dal front-end un messaggio in formato *JSON*. L'esempio mostrato nel Listato 6.19 si riferisce al primo messaggio inviato dall'interfaccia al framework: nel momento in cui l'utente, identificato dall'ID "user", clicca sull'icona del chatbot, il front-end invia un messaggio costituito dall'ID e dall'intento `saluto`.

```
{
"sender": "user",
"message": "/saluto"
}
```

**Listato 6.19:** Esempio di messaggio JSON

Quello che si vuole ottenere è l'integrazione di metadati nel pacchetto JSON. Nell'esempio mostrato nel Listato 6.20 sono presenti i metadati necessari per lo sviluppo del presente ambito. In particolare, il `customer_id` consiste in un ID univoco tramite il quale l'azienda è in grado di identificare il cliente e, di conseguenza, i contratti attivi; il `displayname`, invece, è il nome che l'assistente deve usare per rivolgersi all'utente.

```
{
    "message":"/saluto",
    "sender":"user",
    "metadata" : {
        "customerid" : "75",
        "display_name" : "Leonardo" } }
```

**Listato 6.20:** Messaggio JSON che si vuole gestire

Per permettere a RASA di ricevere e accedere ai metadati, è stato necessario effettuare una modifica ad un file di configurazione del framework creando, così, un custom channel. Esso deriva dal file `/rasa/core/channels/rest.py` in cui sono state applicate delle modifiche in accordo con quanto scritto nella documentazione ufficiale di RASA. I metadati saranno successivamente letti attraverso una funzione definita nel file `actions.py`, mostrata nel Listato 6.21.

```
def get_meta_from_tracker(tracker):
    try:
        events = tracker.current_state()['events']
        user_events = []
        for e in events:
            if e['event'] == 'user':
                user_events.append(e)
        dict_user = json.loads(user_events[-1]['metadata'])['metadata']
        if dict_user is not None:
            return json.loads(dict_user.replace("'", '"'))
        else:
            return None
    except:
        return None
```

**Listato 6.21:** Funzione per l'estrazione dei metadati in un dizionario

### 6.2.2 Config

I componenti che costituiscono la pipeline di NLU sono elencati nel Listato 6.22.

```
pipeline:
  - name: SpacyNLP
    model: it_core_news_md
  - name: SpacyTokenizer
  - name: SpacyFeaturizer
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: "char_wb"
    min_ngram: 1
    max_ngram: 4
  - name: DIETClassifier
    epochs: 100
  - name: ResponseSelector
    epochs: 100
    retrieval_intent: chitchat
```

```
  - name: FallbackClassifier
    threshold: 0.8
```

**Listato 6.22:** Pipeline di NLU

Nel seguito vengono descritti gli elementi non presenti nella pipeline relativa all'ambito promozionale, ovvero:

- `SpacyNLP`: è una libreria gratuita e open-source in Python per eseguire Natural Language Processing. Il modello `it_core_news_md` è un modello linguistico pre-addestrato che può essere inteso come un dizionario della lingua italiana.
- Utilizzando `Spacy`, è possibile impiegare il *Tokenizer* e il *Featurizer* propri della libreria.

Per quanto riguarda le *Dialogue Policy*, esse sono le medesime dell'ambito promozionale.

### 6.2.3 Domain e NLU

La lista degli intenti, delle entità e degli slot è mostrata nel Listato 6.23.

```
intents:
- chitchat:
    is_retrieval_intent: true
- saluto
- cosa_approfondire
- scelta_pratica
- negazione
- affermazione
- addio_pos
- addio_neg
- maggiori_informazioni
- segnalazione
- problematiche
- altro
- info_o_prob
- contatta_xxx

entities:
- scope
- customer_id
- pratica

slots:
  pratica:
    type: rasa.shared.core.slots.TextSlot
    initial_value: null
    auto_fill: true
    influence_conversation: false
  segnalazione:
    type: rasa.shared.core.slots.TextSlot
    initial_value: null
    auto_fill: true
    influence_conversation: false
```

**Listato 6.23:** Intenti, entità e slot

Nel Listato 6.24 venono definiti i nomi delle cinque custom action e viene istanziata una form. Quest'ultima risulta utile per raccogliere una segnalazione da parte dell'utente da inoltrare al supporto.

```
actions:
- lista_pratiche
- analisi_pratica
- maggiori_informazioni
- saluto
- segnalazione

forms:
  segnalazione_form:
    required_slots:
      segnalazione:
        - type: from_text
```

**Listato 6.24:** Custom action e form

P er quanto riguarda il contenuto del file `nlu.yml`, esso viene omesso perché contenente solo esempi per gli intenti di tipo `chitchat`, già descritti nella sezione precedente. Infatti, la conversazione è ugualmente guidata attraverso pulsanti; gli input testuali dell'utente sono gestiti soltanto all'interno del contesto *chitchat*. Come avvenuto per l'ambito precedente, nonostante le risposte siano contenute nel file `domain.yml`, esse sono descritte seguendo il flusso conversazionale.

### 6.2.4 Rules, Responses e Custom actions

Di default, il chatbot in ascolto alla prima interazione è quello relativo all'ambito promozionale. Se l'utente preme il pulsante *Assistenza*, viene inviato a RASA il seguente payload: `/cosa_approfondire{{"scope":""}}`. Tale messaggio viene intercettato dall'interfaccia che, ad uno *scope* vuoto, mette in ascolto il chatbot che gestisce l'ambito customer care. Entrambi i chatbot, quindi, sono sempre in ascolto. È il valore che assume la variabile `scope` a determinare a quale assistente inviare i messaggi.

L'intento `cosa_approfondire` attiva la regola mostrata nel Listato 6.25.

```
- rule: cosa approfondire
  steps:
    - intent: cosa_approfondire
    - action: saluto
```

**Listato 6.25:** Regola per introdurre l'utente all'ambito customer care

La custom action `saluto` ha il compito di estrarre i metadati. Se essi non vengono trovati, significa che l'utente non è autenticato; l'assistente chiederà all'utente di accedere all'area clienti, dando indicazioni su come fare. Se l'utente, invece, ha effettuato correttamente il login, viene eseguita la porzione di codice mostrata nel Listato 6.26, in cui si permette di scegliere tra due opzioni: indagare i pagamenti o "Altro".

```
button = [{"title": 'Pagamenti Conto Energia',
                    "payload": '/scelta_pratica'},
                   {"title": 'Altro', "payload": '/altro'}]
dispatcher.utter_message("Ciao "+displayname+". In questa sezione posso
    ↪ fornirti servizi di assistenza. Ad esempio, cercare problematiche
    ↪ relative ai <b>Pagamenti del Conto Energia</b>.", buttons = button)
```

**Listato 6.26:** Capacità dell'assistente nell'ambito customer care

Nel caso in cui l'utente scelga "Altro", l'intento viene catturato da una regola apposita e viene inviatao in output la risposta contenuta nel Listato 6.27.

```
utter_altro:
- buttons:
```

```
- payload: /scelta_pratica
  title: Pagamenti Conto Energia
text: |
  Per altro tipo di aiuto puoi contattare direttamente il supporto di XXX
     ↪ a questa <a href='https://xxx' target='_blank'>pagina</a>.
  Se vuoi posso continuare ad aiutarti. In questa sezione posso fornirti
     ↪ servizi di assistenza. Ad esempio, cercare problematiche relative
     ↪ ai pagamenti del Conto Energia.
```

**Listato 6.27:** Risposta al pulsante "Altro"

L'intento `scelta_pratica`, che costituisce il payload del pulsante "Pagamenti conto energia", attiva una nuova custom action (chiamata `lista_pratiche`) tramite una regola ad hoc. Tale custom action effettua una serie di operazioni, ovvero:

- Viene effettuato un controllo sul *customer_id* estratto sui metadati. Se l'ID è nullo, si indirizza l'utente alla pagina del supporto. Altrimenti, si effettua una richiesta di tipo GET ad un modulo che, dato un `customer_id`, restituisce la lista delle pratiche associate.
- Viene successivamente effettuato un controllo sulla lunghezza della lista, in particolare:
  - Se la lunghezza della lista è vuota, si comunica all'utente che non risultano attualmente attive pratiche gestite dall'azienda e si indirizza l'utente verso la pagina del supporto.
  - Se la lista ha una lunghezza pari a uno, viene chiesta all'utente una conferma che la pratica ritornata dal modulo è effettivamente quella che egli vuole indagare. Se l'utente conferma, viene generato un payload con intento `/info_o_prob` avente come entità il numero della pratica. Tale entità andrà a "valorizzare" l'omonimo slot. Se, invece, l'utente contradice l'assistente, quest'ultimo richiede di contattare un agente umano.
  - Se la lista delle pratiche ha una lunghezza maggiore di uno, si comunica all'utente che risultano attive più pratiche associate al suo ID. Tramite dei pulsanti, si permette all'utente di opzionare la pratica di interesse. I pulsanti sono accomunati da un payload con l'intento `/info_o_prob`, ma distinti dall'entità che verrà "valorizzata" con il numero della pratica selezionato.

La regola riportata nel Listato 6.28 evidenzia come l'intento `/info_o_prob` valorizzi l'entità e lo slot `pratica`. Inoltre, essa impone che, al verificarsi di tale intento, venga predetta come prossima azione la risposta `utter_info_o_prob`.

```
- rule: scelta tra info aggiuntive o analisi problematiche
  steps:
    - intent: info_o_prob
      entities:
      - pratica: x
    - slot_was_set:
      - pratica: x
    - action: utter_info_o_prob
```

**Listato 6.28:** Regola a seguito della scelta della pratica

Tale risposta richiede all'utente come vuole che il chatbot lo assista sulla pratica selezionata fornendo due opzioni:

- *"Vorrei maggiori informazioni"*: tale pulsante attiva una custom action che esegue una richiesta di tipo GET ad un secondo modulo. La risposta del modulo consiste in un

insieme di informazioni relative alla pratica che vengono formattate in un elenco puntato e restituite all'utente. A questo punto, tramite due ulteriori pulsanti, si permette all'utente di contattare il supporto oppure di indagare la presenza di problemi nella pratica opzionata.

- *"Ho un problema"*: tale pulsante attiva un'ulteriore custom action. Essa effettua una richiesta di tipo GET a un terzo (e ultimo modulo). Tale modulo è costituito dal modello di Machine Learning descritto nella Sezione 4.3.1 e restituisce l'eventuale problematica associata al numero della pratica precedentemente selezionato. La custom action effettua una mappatura tra l'output del modello e le risposte che il cliente ha previsto per ogni tipologia di problematica. All'utente viene ritornato, quindi, un testo composto da una lista di problematiche relative alla pratica (o avverte l'utente che il sistema non ha rilevato nessun problema).

La seconda opzione genera quattro ulteriori pulsanti:

- "*Lista pratiche*": esegue nuovamente la custom action `scelta_pratiche` per permettere all'utente di opzionare una pratica diversa da quella precedentemente selezionata (nel caso ne avesse più di una attiva).
- "*Vorrei maggiori informazioni*": esegue la custom action che genera l'elenco di proprietà della pratica attualmente in esame.
- "*Ho risolto*" e "*Segnalazione*", il cui effetto di entrambi è descritto nel seguito.

Relativamente al pulsante "*Ho risolto*", l'intento che ne costituisce il payload attiva la risposta `utter_feedback`, riportata nel Listato 6.29. Tale risposta ha l'obiettivo di richiedere all'utente un feedback (raccolto dal front-end) che valuti l'intera conversazione sostenuta con l'assistente.

```
utter_feedback:
- buttons:
- payload: /addio_pos{{"feedback":"positivo"}}
  title: Positiva
- payload: /addio_neg{{"feedback":"negativo"}}
  title: Negativa
text: Come valuteresti la conversazione?
```

**Listato 6.29:** Risposta per la richiesta di feedback

Gli intenti `addio_pos` e `addio_neg`, tramite due specifiche regole, generano, rispettivamente, le risposte mostrate nel Listato 6.30. Entrambe le risposte, a seguito di una formulazione di un saluto, permettono all'utente di ricominciare la conversazione con la scelta tra i due ambiti.

```
utter_addio_pos:
- buttons:
  - payload: /cosa_approfondire{{"scope":"on"}}
    title: Servizi
  - payload: /cosa_approfondire{{"scope":""}}
    title: Assistenza
  text: Grazie, sono qui se hai bisogno di me! Ti posso aiutare ad
    ↪ avere informazioni su XXX, i suoi servizi, oppure supportarti
    ↪ se hai problemi nei pagamenti del servizio Conto Energia.
utter_addio_neg:
- buttons:
  - payload: /cosa_approfondire{{"scope":"on"}}
```

```
      title: Servizi
    - payload: /cosa_approfondire{{"scope":""}}
      title: Assistenza
    text: Mi dispiace, cerchero' di fare meglio la prossima volta! Sono
      ↪ qui se hai bisogno di me.
```

**Listato 6.30:** Risposte per il fine conversazione

Per quanto riguarda, invece, il pulsante "*Segnalazione*", l'omonimo intento che ne costituisce il payload attiva la form riportata nel Listato 6.31.

```
- rule: Activate form
  steps:
  - intent: segnalazione
  - action: utter_messaggio
  - action: segnalazione_form
  - active_loop: segnalazione_form

- rule: Submit form
  condition:
  - active_loop: segnalazione_form
  steps:
  # La form viene disattivata
  - action: segnalazione_form
  - active_loop: null
  - slot_was_set:
    - requested_slot: null
  # Lista di azioni da eseguire a seguito della disattivazione
  - action: segnalazione
  - action: utter_segnalazione
```

**Listato 6.31:** Form per la gestione della segnalazione

Procedendo con ordine, la risposta `utter_messaggio` richiede all'utente di inserire nella casella di testo dell'assistente la segnalazione che vuole inoltrare al supporto. Attraverso le istruzioni `action: segnalazione_form` e `active_loop: segnalazione_form` viene attivata la form definita nel *Domain* che "valorizza" lo slot `segnalazione` con ciò che l'utente scrive. La seconda regola, `submit form`, esegue alcune istruzioni per la disattivazione della form. L'azione `segnalazione` è una custom action che genera un ticket in una base di dati, includendo, oltre al testo della segnalazione, il `customer_id` dell'utente, il numero della pratica e il `sender_id`. Infine, la risposta `utter_segnalazione` conferma la creazione del ticket e, tramite due pulsanti, permette di ricominciare la conversazione con la scelta degli ambiti.

CAPITOLO 7

# SWOT Analysis

*In questo capitolo si introduce la SWOT Analysis, spiegandone il procedimento e le potenzialità. Successivamente tale analisi verrà effettuata nel presente progetto.*

## 7.1 Introduzione

L'analisi SWOT (Strengths, Weaknesses, Opportunities, e Threats) è un framework usato per valutare rispettivamente i punti di forza, le debolezze, le opportunità e le minacce di un nuovo progetto, azienda o iniziativa. L'analisi è progettata per facilitare la valutazione dei punti di forza e debolezza in modo realistico, basato sui fatti e guidato dai dati. Come mostrato in Figura 7.1, è possibile etichettare gli elementi che compongono l'analisi con due categorie: fattori utili o dannosi, fattori interni o esterni all'azienda.

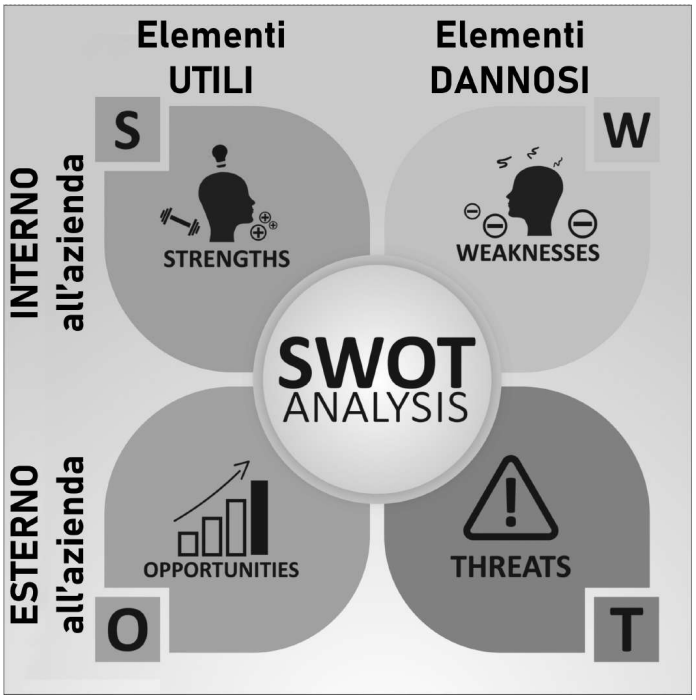


**Figura 7.1:** Matrice SWOT

Gli elementi che costituiscono l'analisi sono:

- *Strenghts*: sono gli aspetti in cui il progetto eccelle e cosa lo distingue dalla concorrenza. Sono, quindi, quegli elementi che facilitano il conseguimento dell'obiettivo di progetto. Ad esempio, i punti di forza possono essere le risorse a disposizione o, in generale, tutto ciò che costituisce il vantaggio competitivo.
- *Weaknesses*: sono gli aspetti che non permettono al progetto di avere performance ottime: sono, cioè, aree dove possono essere fatti miglioramenti. Ad esempio, le debolezze possono essere date da una mancanza di risorse, da ciò che è ancora migliorabile o da qualche componente che sta attualmente sottoperformando. In generale, ci si riferisce a tutto ciò che può essere considerato dannoso rispetto al raggiungimento dell'obiettivo.
- *Opportunities*: sono i fattori esterni favorevoli che potrebbero fornire un vantaggio competitivo. Per esempio, il rilascio di nuove funzionalità di un framework o l'apertura di un nuovo mercato dove è possibile espandersi possono dare vita a opportunità.
- *Threats*: sono fattori esterni che potrebbero danneggiare il progetto o l'organizzazione. Ad esempio, le minacce possono essere costituite da nuove regolamentazioni o leggi che limitano l'uso del prodotto frutto del progetto, oppure una crescente competività dovuta a un interessamento delle tematiche affrontate nel progetto da parte di aziende concorrenti.

## 7.2 Analisi SWOT sul chatbot Gessi

Si andranno, ora, a valutare gli elementi descritti precedentemente in relazione al presente progetto.

### 7.2.1 Strenghts

I punti di forza del progetto sono molteplici. Essendo altamente sperimentale, la Proof Of Concept (POC) realizzata ha permesso al team di progetto di fare uso di tecnologie e metodi molto innovativi. Il presente progetto è stato, quindi, fonte di un grande accrescimento del *know-how* aziendale. Riguardo agli aspetti più tecnici, il progetto offre un'architettura altamente scalabile in entrambi gli ambiti. L'ambito promozionale, infatti, integra all'interno del sistema una componente che effettua lo scraping del sito. Tale funzione permette l'aggiornamento delle informazioni sul sito web, l'inserimento, la cancellazione o la modifica della documentazione e della modulistica, l'aggiunta di documenti PDF etc. senza intaccare l'efficienza dell'algoritmo di neural search. Per quanto riguarda l'ambito customer care, essendo il modello di inferenza un albero di decisione, esso gode di tempi di addestramento relativamente brevi. Ciò permette, all'arrivo di nuovi dati aggiornati, un rapido addestramento del modello. Un ulteriore punto di forza è dato dai costi di produzione contenuti. Essendo il framework e i modelli utilizzati open source e computazionalmente non molto onerosi non è risultato necessario l'acquisto di licenze o calcolatori ad-hoc. Il costo del chatbot è composto, quindi, soltanto dai costi di progettazione, sviluppo e manutenzione dello stesso. Infine, è possibile indicare come un punto di forza la capacità del chatbot di migliorarsi nel tempo. La versione di implementazione (RASA 2.8) è, infatti, compatibile con il tool *Rasa X*, pensato appositamente per uno sviluppo di tipo *Conversational-Driven*. In breve, *Rasa X* offre un'interfaccia per l'accesso alle conversazioni sostenute dagli utenti con l'assistente, permettendo l'annotazione di nuovi intenti, entità o flussi conversazionali veri e propri. Tali annotazioni costituiranno un aggiornamento dei dati di training di RASA, il quale sarà, quindi, in grado di adattarsi alle esigenze degli utenti.

### 7.2.2 Weaknesses

Le debolezze del progetto sono principalmente causate da un utilizzo di tecnologie ancora non perfettamente mature. Infatti, Haystack non sempre fornisce risposte esattamente coerenti con le domande poste dagli utenti. Inoltre, la componente *AI-based* di RASA non è stata valorizzata a pieno. Infatti, il file di addestramento del modello che esegue Natural Language Understanding (NLU) per la classificazione degli intenti e l'estrazione delle entità è stato lasciato volutamente con pochi contenuti. Tale scelta, decisa in fase di progettazione per consentire una conversazione guidata tramite pulsanti, può essere limitante nel momento in cui l'utente si aspetta un'esperienza più *human-like*. Un'ulteriore debolezza consiste nella fruibilità del servizio solamente in lingua italiana, nonostante il sito sia consultabile anche in lingua inglese. Tale aspetto è sicuramente fonte di aggiornamenti futuri ma, al momento, rappresentano una limitazione nell'utilizzo del chatbot. Infine, l'ultima debolezza individuata è relativa ai dati forniti per l'addestramento del modello di inferenza nell'ambito customer care. Infatti, come descritto nella Sezione 4.3, il dataset presenta classi fortemente sbilanciate. Nonostante i test, a seguito di un bilanciamento tramite tecniche di oversampling, abbiano fornito risultati ottimi, tale caratteristica può portare ad una diminuzione della robustezza del modello, principalmente per alcuni casi particolari.

### 7.2.3 Opportunities

Le opportunità relative al progetto sono principalmente attinenti all'immagine dell'azienda Filippetti rispetto alle organizzazioni concorrenti. Infatti, lo sviluppo di un chatbot che fa utilizzo di tecnologie innovative comporta un importante accrescimento della visibilità dell'organizzazione. Inoltre, il *know-how* aziendale acquisito sarà facilmente spendibile in progetti futuri essendo i chatbot e, più in generale, le tecniche di Intelligenza Artificiale e Machine Learning sempre più richiesti in ambito imprenditoriale. In ultimo, essendo il cliente un'azienda il cui obiettivo è la crescita della sostenibilità energetica in Italia, le risorse precedentemente usate per il supporto clienti (che con l'integrazione del chatbot diventeranno superflue) potranno essere concentrate nella ricerca di nuove tecnologie sul settore rinnovabile o di metodologie per l'efficientamento energetico.

### 7.2.4 Threats

Per quanto riguarda le possibili minacce, la più immediata concerne in un avvicinamento delle aziende concorrenti alle tematiche affrontate nel progetto. L'immaturità del framework e delle tecnologie utilizzate, infatti, potrebbero inizialmente far desistere le aziende a investire in tali aspetti. Ma avendone dimostrato le potenzialità e i punti di forza, è probabile che un numero sempre più crescente di organizzazioni inizi ad approcciarsi a queste tematiche. Infine, l'introduzione di un chatbot impatterà nel comportamento dell'utente. È importante conoscere chi è l'utente finale e quanto egli è incline al cambiamento. Se l'utente medio è più "conservatore", allora il chatbot potrebbe essere utilizzato in modo ridotto e la mole di richieste al service desk rimarrebbe invariata.

# Conclusione

Questa tesi ha esposto i metodi per la progettazione e lo sviluppo di un chatbot per la creazione di un canale alternativo rispetto agli agenti umani del contact center. Lo scopo del chatbot è duplice: si vuole facilitare la consultazione del contenuto del sito web e della modulistica allegata e, contemporaneamente, permettere all'utente di indagare la presenza di problematiche relative ai pagamenti. In fase di progettazione, si sono individuate le componenti necessarie al raggiungimento di tali obiettivi. In particolare, si è optato per la realizzazione di due sistemi distinti accumunati da uno stesso cuore: il chatbot. Gli elementi di contorno, fondamentali per l'esecuzione dei suddetti task, sono costituiti principalmente da due modelli di Intelligenza Artificiale e Machine Learning: il primo ha il compito di trovare una porzione di testo in un dataset di tipo testuale il più coerente possibile rispetto a una domanda posta dall'utente e, il secondo, dato un'insieme di informazioni relative ad una pratica di un utente, restituisce zero, una o più problematiche associate alla pratica, basandosi su un dataset di addestramento. Lo scopo del chatbot è quello di interagire con l'utente per estrarre i dati necessari all'esecuzione dei task e, allo stesso tempo, comunicare con i modelli per restituire all'utente le informazioni richieste. Sono stati progettati dei flussi conversazionali flessibili ed esaustivi, in cui l'utente viene guidato attraverso la conversazione attraverso dei pulsanti. La scelta dell'utilizzo di pulsanti permette di chiarire all'utente quali sono le funzionalità del chatbot e mantenerlo all'interno degli *happy-path*. Il chatbot è stato sviluppato in RASA. Tale framework offre un insieme di funzionalità che hanno reso RASA completo rispetto alle esigenze del cliente.

Benché fosse un progetto sperimentale, i risultati ottenuti hanno pienamente soddisfatto il committente. Nonostante ciò, sono previsti sviluppi futuri per l'integrazione di funzionalità e miglioramenti del sistema complessivo. In particolare, per l'ambito promozionale è possibile migliorare l'algoritmo che effettua lo scraping del sito web e dei PDF allegati. Infatti, attualmente esso ha difficoltà nel tradurre in testo informazioni strutturate in tabelle e ignora le immagini. Inoltre, uno studio più approfondito del modello di Neural Search, può portare a risposte più esatte. Nel dettaglio, Haystack ha integrato un dizionario pre-addestrato della lingua italiana. Generare un dizionario specifico nel dominio caratteristico di applicazione e integrarlo nel modello porterà, di certo, ad un miglioramento dei risultati. Invece, per quanto riguarda il chatbot, esso dovrà essere capace di sostenere conversazioni anche in lingua inglese. Essendo il sito web fruibile anche in tale lingua, il front-end potrebbe integrare tra i metadati gestiti dal custom channel anche la configurazione linguistica impostata dall'utente nell'interfaccia. Tale informazione sarà gestita all'interno di RASA dove saranno implementati gli stessi flussi conversazionali presentati in questa tesi, ma in lingua inglese.

# Bibliografia


ADAMOPOULOU, E. (2020), «Chatbots: History, technology, and applications», Rap. tecn., ScienceDirect.

ADAMOPOULOU, E. e MOUSSIADES, L. (2020), «An overview of chatbot technology», in «IFIP International Conference on Artificial Intelligence Applications and Innovations», p. 373–383, Springer.

CHAVES, A. P. e GEROSA, M. A. (2021), «How should my chatbot interact? A survey on social characteristics in human–chatbot interaction design», *International Journal of Human–Computer Interaction*, vol. 37 (8), p. 729–758.

CUI, L., HUANG, S., WEI, F., TAN, C., DUAN, C. e ZHOU, M. (2017), «Superagent: A customer service chatbot for e-commerce websites», in «Proceedings of ACL 2017, system demonstrations», p. 97–102.

DAVE GERHARDT, D. C. (2019), *Conversational Marketing: How the World's Fastest Growing Companies Use Chatbots to Generate Leads 24/7/365 (And How You Can Too)*, John Wiley & Sons Inc.

KEMPT, H. (2021), *Chatbots and the Domestication of AI: A Relational Approach*, palgrave Macmillan.

LAM, K. N., LE, N. N. e KALITA, J. (2020), «Building a Chatbot on a Closed Domain using RASA», in «Proceedings of the 4th International Conference on Natural Language Processing and Information Retrieval», p. 144–148.

LOKMAN, A. S. e AMEEDEEN, M. A. (2018), «Modern chatbot systems: A technical review», in «Proceedings of the future technologies conference», p. 1012–1023, Springer.

MAMATHA, M. e OTHERS (2021), «Chatbot for e-commerce assistance: based on RASA», *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 12 (11), p. 6173–6179.

NAGARHALLI, T. P., VAZE, V. e RANA, N. (2020), «A review of current trends in the development of chatbot systems», in «2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)», p. 706–710, IEEE.

NGAI, E. W., LEE, M. C., LUO, M., CHAN, P. S. e LIANG, T. (2021), «An intelligent knowledge-based chatbot for customer service», *Electronic Commerce Research and Applications*, vol. 50, p. 101 098.

PARKER, P. P. M. (2022), *The 2023-2028 World Outlook for Chatbots*, iCON Group International.

RAJ, S. (2018), *Building Chatbots with Python: Using Natural Language Processing and Machine Learning*, Apress.

RAPP, A., CURTI, L. e BOLDI, A. (2021), «The human side of human-chatbot interaction: A systematic literature review of ten years of research on text-based chatbots», *International Journal of Human-Computer Studies*, vol. 151, p. 102 630.

RYDNING, J. (2021), «Worldwide Global DataSphere and Global StorageSphere Structured and Unstructured Data Forecast, 2021–2025», Rap. tecn., Global DataSphere. (Cited at page 3)

SEDOC, J., IPPOLITO, D., KIRUBARAJAN, A., THIRANI, J., UNGAR, L. e CALLISON-BURCH, C. (2019), «Chateval: A tool for chatbot evaluation», in «Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics (demonstrations)», p. 60–65.

SHARMA, R. K. e JOSHI, M. (2020), «An analytical study and review of open source chatbot framework, rasa», *International Journal of Engineering Research and*, vol. 9 (06).

SINGH, A. (2019), *Building Chatbots with Python: Using Natural Language Processing and Machine Learning*, apress.

## Siti web consultati

- IBM Cloud Learn Hub – `www.ibm.com/cloud/learn`
- RASA Developer Documentation Portal – `https://rasa.com/docs/rasa/`
- Investopedia – `https://www.investopedia.com/`
- MindTitan– `https://mindtitan.com/resources/`